University of Alberta

File Store Memories

by

Curtis Wickman

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2000

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-82377-6

Canada

University of Alberta

Library Release Form

Name of Author: Curtis Wickman

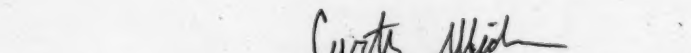
Title of Thesis: File Store Memories

Degree: Master of Science

Year this Degree Granted: 2000

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly, or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



Curtis Wickman
wickman@ieee.org
10155 63 Street
Edmonton, Alberta, Canada
T6A 2M5

Date: July 25, 2000

And still I am learning.
Michelangelo Buonarotti

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled File Store Memories submitted by Curtis Wickman in partial fulfillment of the requirements for the degree of Master of Science.

D S Elliott

Duncan Elliott, Co-Supervisor

Bruce Cockburn, Co-Supervisor

B. Nowrouzian

Behrouz Nowrouzian

M. A. MacGregor

Mike MacGregor

2000-7-25

To Jennifer.

ABSTRACT

When dynamic random access memory (DRAM) is manufactured it must have 100% of the nominal capacity in order to be sold. If there are any errors in the DRAM then it cannot be sold even though a very large portion of it is error-free. This thesis investigates techniques that allow some of the memory that would otherwise be wasted to be used by building file store memories. Some techniques investigated are bad block marking (marking regions of memory as bad) and using error correction codes. These techniques are not limited to DRAM and can also be extended to other memories such as flash memory.

ACKNOWLEDGMENTS

This research was supported by Micronet R&D, MOSAID Technologies Inc., the Canadian Microelectronics Corporation, and the University of Alberta. I would like to give special thanks to supervisors Duncan Elliott and Bruce Cockburn for all the help and directions that they gave during this thesis.

TABLE OF CONTENTS

Chapter 1: Introduction **1**

1.0.1. Motivation 1

1.1 Background 2

1.1.1. The DRAM Market 2

1.1.2. Memory Architecture 2

1.1.2.1 DRAM Array Architecture 3

1.1.2.2 Traditional DRAM Row and Column Redundancy 3

1.1.2.3 Flash Array Architecture 3

1.2 Cost Models 4

1.3 Application Analysis 4

1.3.1. MP3 Player 4

1.3.1.1 Examining Current Consumption for the Storage 5

1.3.1.2 Results 5

1.3.2. Digital Camera 6

1.3.3. Other Applications 7

1.4 Microprocessor versus DRAM Feature Size 8

1.5 Organization of the Thesis 11

Chapter 2: Effects of Conventional Fault Tolerance on Yield **13**

2.1 Synergistic Fault Tolerance 13

2.2 Yield 14

2.3 Modeling Using Equivalent Yield 15

2.4 Hamming Codes in DRAM 15

2.4.1. Hamming Codes 15
2.4.2. Hamming Code Operation 16
2.4.3. Modified Hamming Codes 17
2.4.4. Hamming Codeword Size 19
2.4.5. Area of Hamming Code Circuitry 20
2.4.5.1 Hamming Code Hardware 20
2.4.5.2 Hamming Code Delay 23
2.5 Using ECC to Improve Both Hard and Soft Error Rates 25
2.5.1. Hard Error Correcting Affecting Soft Error Rate 27
2.6 Additional Error Correcting Codes 29
2.7 Redundant Rows and Columns 29
2.7.1. The Cost of Redundancy 30
2.8 Effects of ECC, Redundancy, and ECC & Redundancy on Yield 30
2.8.1. Yield Assuming No ECC or Redundancy 30
2.8.2. Yield Using Only Redundant Columns 31
2.8.3. Yield Using Only Redundant Rows 32
2.8.4. Yield Using Redundant Columns and Redundant Rows 32
2.8.5. Yield Using Only ECC 33
2.8.6. Yield Using Redundant Columns and ECC 33
2.8.7. Yield Using Redundant Columns, Rows, and ECC 34
2.9 Applying Yield Equations to the IBM 16Mb DRAM 34
2.10 1Gb DRAM Example 38
2.10.1. Architecture of a 1Gb DRAM 38
2.10.2. Calculating the Yield of a 1Gb DRAM with Redundancy 39

2.10.3. Equivalent Yield of a 1Gb DRAM with Different ECC Word Sizes....42

2.11 Summary....43

Chapter 3: Bad Block Marking Applied to Memory....45

3.1 Non-Volatile Memory for Storing a List or Bitmap....45

3.1.1. A Negligible Die Overhead Method for Marking Bad Blocks....45

3.2 Bad Block Marking....46

3.2.1. Partially Good Products Method....46

3.2.2. Motivation for Bad Block Marking....47

3.3 Bad Block Marking Techniques....48

3.4 Marking Cells as Bad....48

3.4.1. Bitmap Marking....48

3.4.2. List Marking....49

3.4.3. Combination Marking....55

3.5 Marking Bitlines as Bad....55

3.6 Using Bad Block Marking with DRAM....56

3.7 Applying Bad Block Marking to the IBM 16Mb DRAM....56

3.8 Bad Block Marking Applied to the 1Gb DRAM....59

3.8.1. Bad Block Marking....60

3.8.2. Bad Block Marking with Row and Column Redundancy....60

3.9 Conclusions....62

Chapter 4: Systems Measurements....65

4.1 Hamming Code Performance....65

4.1.1. Measuring the Performance Impact of Off-chip Correction....65

4.1.2. Results of Off-chip Correction 66

4.2 Bad Block Marking Performance 67

4.2.1. Bad Block Marking Performance Setup 67

4.2.2. Bad Block Marking Scheme 67

4.2.3. Results on a 333MHz PII With Disabled Cache 68

4.2.4. Results Using Average Number of Defects 70

4.2.5. Bad Block Marking Random Access Performance 71

4.3 Summary 73

Chapter 5: ECC Implementation and Area 75

5.1 ECC Circuit Area Overhead 77

5.2 Summary 78

Chapter 6: Conclusion 79

6.1 Summary of Results 79

6.2 Future Work 80

Bibliography 81

Appendix A: ECC Hardware 87

A.1 (137,128) ECC Hardware 87

Appendix B: Hamming Codes 88

B.1 (22,16) IBM System/3 [26] 88

B.2 (40,32) IBM 8130 [26] 88

B.3 (72,64) IBM 3033 [26] 88

B.4 (72,64) IBM 3081 [25] 88

B.5 (137,128) IBM DRAM [2] 89

LIST OF TABLES

TABLE 1-2. Power Consumption for Various Applications 8

TABLE 2-1. Hamming Code Matrix 16

TABLE 2-2. Modified Hamming Code Check Bits 18

TABLE 2-3. Area of SEC-DED vs. DED 23

TABLE 2-4. Variable Values Using the IBM 16Mb DRAM 34

TABLE 2-5. Average Number of Defects at a 50% Yield / Equivalent Yield 37

TABLE 2-6. Variable Values Using a 1Gb DRAM 38

TABLE 2-7. Average Number of Defects at a 50% Yield / Equivalent Yield 41

TABLE 3-1. 50% Equivalent Yield Points for Different Amounts of Non-Volatile Memory 54

TABLE 3-2. 50% Equivalent Yield Points for BBM and Row, Column Redundancy ... 62

LIST OF FIGURES

FIGURE 1.2. Traditional DRAM Row and Column Redundancy 3

FIGURE 1.3. Power Consumption as a Function of Transfer Rate 8

FIGURE 1.4. DRAM and Microprocessor Feature Size Over Time 11

FIGURE 2.1. IBM Redundancy, ECC, and Redundancy/ECC plots [1] 14

FIGURE 2.2. Effects of Different Codeword Sizes on the Equivalent Yield 19

FIGURE 2.3. Hamming Code Error Detection Hardware [26] 21

FIGURE 2.4. Hamming Code Error Correction Hardware [26] 21

FIGURE 2.5. Comparison of ECC Modified Hamming Code Circuitry Area 22

FIGURE 2.6. SEC-DED ECC Circuit Delay 24

FIGURE 2.7. SED ECC Circuit Delay 25

FIGURE 2.8. How ECC Affects FIT for a 16Mb and a 1Gb DRAM 27

FIGURE 2.9. Acceptable Soft Error Rate Using ECC using 16Mb 28

FIGURE 2.10. Acceptable Soft Error Rate Using ECC using 1Gb 29

FIGURE 2.11. Yield of IBM 16Mb DRAM Using Redundancy and ECC 35

FIGURE 2.12. Equivalent Yield of IBM 16Mb DRAM Using Redundancy and ECC 36

FIGURE 2.13. Equivalent Yield of IBM 16Mb DRAM for Column, Row, Column and Row Redundancy 37

FIGURE 2.14. Yield of 1Gb DRAM using Redundancy and ECC 39

FIGURE 2.15. Equivalent Yield of 1Gb DRAM Chip Using Redundancy and ECC 40

FIGURE 2.16. Equivalent Yield of 1Gb DRAM for Column, Row, Column and Row Redundancy 41

FIGURE 2.17. Average Number of Tolerable Defects at 50% Equivalent Yield 42

FIGURE 3.1. 128Kb of Non-Volatile Memory for Bad Block Marking List Using 1Gb DRAM....51

FIGURE 3.2. 256 Bits of Non-Volatile Memory for Bad Block Marking....52

FIGURE 3.3. Bad Block Marking List Technique with Different Non-volatile Memory Sizes....54

FIGURE 3.4. Equivalent Yield Using Bad Block Marking in a 16Mb DRAM....57

FIGURE 3.5. Equivalent Yield Using Bad Block Marking in a 16Mb DRAM (continued)....58

FIGURE 3.6. Equivalent Yield Using Bad Block Marking, High Defects....59

FIGURE 3.7. Equivalent Yield Using Marking Bad Blocks in a 1Gb DRAM....60

FIGURE 3.8. Equivalent Yield Using Bad Block Marking, Row and Column Redundancy....61

FIGURE 3.9. Equivalent Yield using Bad Block Marking, Row and Column Redundancy (Top portion)....62

FIGURE 3.10. Profit Regions for Manufacturer for BBM and Redundancy....63

FIGURE 4.1. Performance Degradation With Off-Chip Error Correction....66

FIGURE 4.2. Access Time for Normal DRAM and for a Simple Bad Block Marking List....69

FIGURE 4.3. Performance of Bad Block Marking on a 333MHz PII PC....70

FIGURE 4.4. Real Performance of a Bad Block Marking Scheme....71

FIGURE 4.5. Random Access Performance Using BBM....72

FIGURE 5.1. (22,16) Hamming Code Layout....75

FIGURE 5.2. Circuit Size for Detection, and Detection and Correction....76

FIGURE 5.3. Comparison of Area of Standard Cells for Detection, and Detection/ Correction....77

GLOSSARY OF TECHNICAL TERMS

Conventional Yield

The ratio of the number of working dies manufactured to the total number of manufactured dies.

Equivalent Yield — The yield multiplied by the ratio of working data bits over the total number of bits. It is used to compare systems with redundancy and systems without redundancy. It accounts for the extra area required by the redundancy.

File Memory — Memory (Volatile or Non-Volatile) that is designed for access patterns that are large (compared to the memory size) and mostly sequential.

FIT — Failures in Time is a measure of the soft error rate and is defined as the expected number of errors in 10^9 hours.

Flash — A type of non-volatile memory that uses a floating gate to hold a charge that represents a bit value.

Hamming Code — A type of Error Correction Code that creates a code from data written to memory so that Single Error Correction, Double Error Detection can be accomplished.

Hard Error — A hard error is a physical defect. In DRAM a hard error is an error in the fabrication of the memory cell or support circuitry that appears as a memory cell failure. The error is always present when accessing the memory cell.

Modified Hamming Code

A class of Hamming Codes that attempts to minimize the area by minimizing the number of gates required by choosing the lowest weight codewords, and by equalizing the number of inputs per check bit function.

Soft Error — A soft error is defined as a transient error and in DRAM occurs in a memory cell. The error can be caused by various phenomena including gamma radiation, alpha particle radiation, array noise, etc.

Synergistic Fault Tolerance

When the effect of two redundant fault tolerant techniques used together is greater than the sum of the two techniques.

LIST OF ACRONYMS

BBM	Bad Block Marking
CFII	Compact Flash type 2. A standard that specifies an interface for personal media cards that are interchangeable. CFII cards provide storage or I/O functionality. CFII cards that provide storage are built using flash memory or microdrives.
DRAM	Dynamic Random-Access Memory
ECC	Error Correction Code
EDO	Extended Data Out
EEPROM	Electrically Erasable and Programmable Read-Only Memory
EPROM	Electrically Programmable Read-Only Memory
MLDRAM	Multi-Level DRAM
MP3	Mpeg Layer 3 Audio file
MPEG	Motion Picture Experts Group
MTBF	Mean Time Between Failures
PDA	Personal Digital Assistant
RAM	Random Access Memory
ROM	Read Only Memory
SDRAM	Synchronous DRAM
SEC-DED	Single Error Correction, Double Error Detection
SED	Single Error Detection
SRAM	Static Random Access Memory

Chapter 1: Introduction

Conventional digital systems obtain file storage services using magnetic and optical disks, which provide high capacity storage at the cost of relatively high latency for random accesses and relatively low reliability and ruggedness compared to solid-state components. The advantage of such storage is the low initial and incremental cost per byte. The proposed techniques for solid-state memory will not be cheaper per byte than conventional magnetic storage in the foreseeable future, but solid state memory does give greater reliability and shorter access time compared to magnetic storage. High-density semiconductor file memories offer a storage option that combines high capacity with the superior latency and reliability of solid-state memory. File memories address several strategic driving applications including multimedia applications, video-on-demand services, hypertext servers, digital voice and image storage, advanced personal communications systems (PCS), diskless network computers, set-top boxes, broadband digital communications, and solid state disks for wearable computers [6]. In this thesis file store memory techniques will be investigated in order to reduce semiconductor memory cost by exploiting yield enhancement techniques and by relaxing the traditional random access constraints of memory.

This thesis revisits an old idea with new objectives to propose what we call file memory, in which the requirements of modern memory are relaxed in order to increase the equivalent yield and decrease the average cost per working bit. Specifically, file memory is designed to provide competitive page mode access performance, as opposed to fast random-access. Conventional bit-level or word-level random access is not essential for memory devices that lie lower down in the memory hierarchy, devices which typically operate in block-oriented sequential access mode. Also, many data-intensive applications access data in larger-sized blocks for which the overhead of the initial access can be amortized over the large number of data bits in the block, provided that the average data bandwidth is sufficiently high. The second assumption, that 100% of the nominal memory capacity is available in all parts, is not required of magnetic mass storage devices. System software or device firmware copes with bad sectors or blocks by keeping track of them and avoiding them by means of a software mapping from nominal memory addresses to working physical locations on the imperfect storage medium. Likewise, in file memory there is a possibility that bad blocks can be tolerated by recording the location of all bad blocks and then, possibly in cooperation with system software, redirecting memory accesses to avoid bad blocks.

1.0.1. Motivation

File store memories are intended to lower the cost per bit. Large, multimegabit-scale semiconductor memories can be made cheaper if we remove the conventional constraints that dictate that the memory space must be continuous and one hundred percent functional and that randomly selected storage locations have equal access times. File store memories should result in a higher profit for the manufacture and a lower price to the consumer. Although DRAM prototypes of 256Mb through 4Gb capacity have already been reported

by several manufacturers, the date of their economical mass production is unclear due to the difficulty of manufacturing 100% functional chips. DRAM chips have operated under the following random access constraints: (1) random-access is fast, (2) 100% of the nominal capacity of working bits is present, (3) the size of the memory is a power of 2, and (4) there is a fixed access time. If we relax these requirements, we can improve the profit margins of manufacturing.

1.1 Background

There are two main classes of memory: volatile and non-volatile. Volatile memories, such as DRAM and SRAM, lose their contents when power is lost. Non-volatile solid state memories, such as flash memory, ROM, E^2PROM, and EPROM, retain their data when power is lost. Hard drives are a non-volatile storage medium because they do not require power to maintain their data. Solid state non-volatile memories have several advantages over hard drives such as reliability, bandwidth, and latency. Hard drives are very low cost per byte and solid state components will likely never be cheaper per byte. Of all the solid state memories (volatile and non-volatile) described above, the least expensive is DRAM because of its dense architecture (each cell requires only one transistor and one capacitor).

1.1.1. The DRAM Market

DRAM manufacturing is a high-volume, low profit margin commodity business where every cent that the manufacturer can save is significant. Very large capacity 1Gb and 4Gb DRAM prototypes have been reported. However, it is unclear how long it will take for this part class to be less expensive per bit compared to the currently predominant 64Mb and 256Mb DRAM chips. It is the goal of this research to investigate options for bringing the very large capacity DRAMs or other memories to market sooner and with a lower cost per working bit.

1.1.2. Memory Architecture

One very important constraint for currently manufactured DRAM chips and flash memories is that they need to appear to have a continuous address space of fully functioning cells. Being continuous means that every addressable location within the memory array can be written and read. Not only does the memory need to be continuous, it also needs to provide full capacity. In a sellable DRAM every addressable location must not only work but also have a refresh interval that meets the expected specification. In current DRAM manufacturing practice, redundant bitlines and redundant word lines are used to replace any faults within the array. This allows the manufacturer to repair faulty DRAM chips so that they appear to the users to provide a full capacity, continuous space of functioning cells. If the redundancy provided in manufacturing is insufficient for repairing the defects in the DRAM array, then the die cannot be sold and becomes waste.

1.1.2.1 DRAM Array Architecture

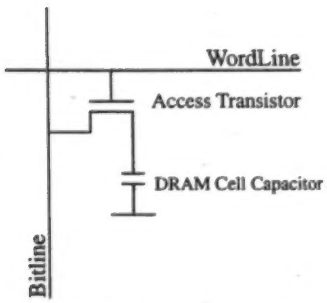


FIGURE 1.1. Basic DRAM Cell

As mentioned earlier, a DRAM cell stores each bit as a charge on a capacitor. The basic DRAM cell is shown to the left in Figure 1.1. The wordline is asserted to turn on the access transistor and the DRAM cell capacitor charge is then shared with the bitline. The slight change in bitline voltage is then sensed by a sense amplifier (not shown) to determine what bit value was in the cell. The reads are destructive because the charge encoded signal is greatly diluted. Therefore, the sensed value must be written back or "restored" to the cell before the cell can be returned to data storage mode.

1.1.2.2 Traditional DRAM Row and Column Redundancy

DRAM's currently use row and column redundancy. When a DRAM is built there are space rows and columns added that can be electronically swapped into the array so that defective elements in the array can be mapped around. Figure 1.2 shows a simplified version of the row and column redundancy for DRAM.

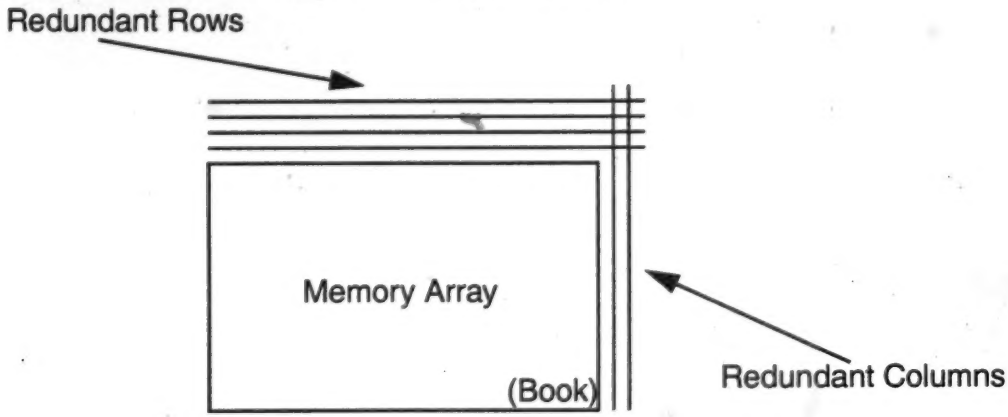


FIGURE 1.2. Traditional DRAM Row and Column Redundancy

1.1.2.3 Flash Array Architecture

As mentioned earlier, flash memory is a type of memory that is non-volatile. This means that the value stored in the flash cell is not destroyed when the power is lost. Flash memory cells use a floating gate to store a charge that represents a coded bit value. One other drawback of flash memory, apart from a slow write time, is that a limited number of writes are possible.

1.2 Cost Models

Cost models help clarify the economic trade-offs present in alternative design choices. In DRAM design, cost models have been used to guide the selection of page sizes and redundancy configurations that permit permanent on-chip repair.

In Chapter 2, cost models appropriate for large file memory DRAM chips are presented that exploit various combinations of error-correcting codes, redundant elements, and bad block marking to reduce the effective cost per functional unit. The relaxed requirements of file memory make it possible to reconsider known fault tolerance methods and to apply them in combination in new ways. Our proposed cost models will guide the selection of the most economical file memory architectures.

1.3 Application Analysis

An important part of investigating file memory is determining what applications file memory could be used for. For example, the 3Com Palm Pilot is a PDA (Personal Digital Assistant) which uses EDO DRAM that has a low power self-refresh mode to maintain the DRAM contents. A possible application for file memory would be to replace the flash memory in an MPEG audio layer three (MP3) player with DRAM. Another possible application would be to use file memory instead of flash cards for digital cameras. There are perhaps more obvious potential advantages to using flash memory in a digital camera than in a MP3 player. The advantages are listed below. The file store techniques that are examined in this thesis apply mostly to DRAM. They do not, however, need to be restricted to this type of memory and could be extended to non-volatile memory types such as flash memory. One of the advantages to using DRAM over flash memory is that the cost of DRAM is significantly lower than the cost of flash. Also, writing to DRAM is much faster. The one down side to using DRAM instead of flash memory is the requirement for battery backup and sufficiently frequent refreshing.

1.3.1. MP3 Player

Currently MP3 players use flash memory as one storage medium to store MP3-encoded files. Some MP3 players use smart media and others use Compact Flash Type 2 (CFII) microdrives. The Diamond Rio 500 MP3 player, for example, has 64Mb of flash built in and will play music for 13 hours on one AA battery. It retails for approximately $270US. A double A battery delivers approximately 2800mAh with an initial voltage of 1.5V. Assuming that the chips in the Rio run on 3V and use a step-up converter that is 100% efficient, there is approximately 1400mAh available for the player. If it works for 13 hours then it consumes approximately 107.7mA, which is an average power drain of 323.2mW.

1.3.1.1 Examining Current Consumption for the Storage

DRAM

Eight DRAM chips are needed to compare to the 64Mb of flash if an 8MB Samsung EDO DRAM (KM416V4104C) chips (which are compatible with the Palm Pilot) are used. The 64Mb DRAM consumes 350µA when it is in low power self refresh mode, 80mA when it is in EDO mode, and 500µA when it is in standby mode. A more modern 256Mb SDRAM from NEC will be used in the analysis because of its size and lower power consumption. The NEC part number is µPD45256163. It is a 16 bit output DRAM with 1mA current consumption when it is in low power self refresh mode, 100mA when it is in burst read write mode, 1mA in stand-by mode, and a maximum burst size of 512 16 bit words.

Good quality MP3's are recorded at 44kHz sampling rate and producing raw data at 128Kbps and, after being compressed, take approximately 16KB per second. This is approximately 1MB per minute. The time required to burst transfer an entire page is one transfer at 60ns and 511 transfers at 15ns. This works out to a total time of 7.725µs. There are 16 burst read transfers required per second (in order to get the 16KB per second) for a total of 123.6µs per second. Assuming that, of the two DRAM chips, only one is active at one time and the other is in standby, then the current can be calculated as 1mA for the standby DRAM plus 1.012mA $(123.6\mu s \cdot 100mA + (1 - 123.6\mu s) \cdot 1mA)$ for the active DRAM. The total current becomes 2.012mA with power consumption of 6.64mW. If the DRAM is put into self refresh mode it consumes 2mA and would last approximately 700 hours (i.e. 28 days) on a double A battery.

Flash

Four 128Mbit flash chips are required to make up the 64MB storage. Examining the data sheets for the 28F128J3A (an Intel flash memory http://developer.intel.com/design/flash/datashts/index.htm) gives us a standby current of 50µA, read current of 20mA, and a write current of 60mA. The setup time for a read is 100ns with a page read mode cycle time of 25ns. The flash memory requires 6µs per byte when writing. Assuming that 1KB is accessed at a time, the total time required for the read is 12.875µs. This works out to an approximate current of 0.2mA if only one flash is active at a time. Converting this current to power consumption yields approximately 0.67mW.

1.3.1.2 Results

Table 1-1 below shows the current consumption and the expected battery life when comparing DRAM to flash memory for an average transfer rate of 16KB per second. Equation (1.1) below shows how the expected number of hours can be calculated based on

the current consumption of flash memory and DRAM. Dividing the number of milliampere hours by the current minus the flash plus the DRAM gives.

$$BatteryLife = \frac{1400mAh}{(107.7mA - 0.2mA + 2.0mA)} = 12.8hours \quad (1.1)$$

In Equation (1.1) the 107.7 mA is from Section 1.3.1. The flash memory costs approximately $9 per 32Mb flash chip (all prices given will be in US dollars). This works out to the flash costing approximately $148. The spot price of a 64Mb DRAM as of May 24, 2000 is approximately $6-7, which works out to a cost of about $50 for 64MB of DRAM. Price comparison has been done using 64Mb, and not 256Mb, because it is hard to find quantity prices for 256Mb DRAM.

TABLE 1-1. Analysis of Flash versus DRAM

Memory Component	Off		On		Cost for the memory
	Power	Battery Life	Power	Battery Life	
64Mb Flash Memory	-	Inf	0.67mW	13 hours	$148
64Mb DRAM	6.6mW	700 hours	6.64mW	12.8 hours	$50

When putting this information back into the Diamond Rio 500 MP3 player we can reduce the memory cost of the player by a significant factor of 2/3 by replacing the flash memory with DRAM. For the DRAM version a rechargable battery could be used and the player could be put into a cradle that recharges the battery when it is not in use. The cradle could also allow new songs to be downloaded to the player via a serial, parallel, or Internet connection.

1.3.2. Digital Camera

Another area where flash memory is used is digital cameras. The difference between a digital camera and an MP3 player is that the MP3 player does not write to memory. A digital camera does however require writes to memory. Flash consumes more power on memory writes than on reads (60mA and 20mA, respectively). Another consideration for using DRAM in digital cameras instead of flash is the camera cycle time. The camera must compress the image and then write it to flash before it can take the next picture. Some cameras have large DRAM buffers in the camera that can allow a second picture to be taken only a few seconds after the first picture. The number of images that can be taken using the buffer is limited and is usually no more than two or three images. The flash used in the above MP3 player evaluation requires an effective programming time of 6µs per byte. The Nikon coolpix 990 uncompressed image is approximately 10Mb which would take approximately 63 seconds. This gives us a cycle time of roughly 63 seconds plus exposure time. The DRAM can do a burst write at 25ns per 16 bits, which works out to the 10MB image being written in 0.262 seconds, i.e. approximately 240 times faster than flash memory. This means that using DRAM instead of flash memory in cases where writes

occur may prolong the battery life because flash writes consume more power. Using DRAM would also be cheaper than using an equivalent amount of flash memory. The one downside to using DRAM is that a battery is required so that the DRAM contents are refreshed. But it is likely that the camera will be provided with a long-lasting battery source.

The average required transfer rate is the size of the image taken over the amount of time the camera is on. Assuming that an image taken is about 1MB (after being JPEG compressed) and assuming that the camera would be on for about 30 seconds we get an average transfer rate of approximately 34KB/s.

1.3.3. Other Applications

The analyses of power consumption given for the MP3 player shows that flash memory consumes less power than DRAM but the difference in playtime is not noticeable because the memory consumes much less power than the player. We also know that the power consumption for flash memory depends on what types of accesses are being done (reads or writes). The power that is consumed will depend on the type of memory accesses and on the average data transfer rate. Figure 1.3 plots the power consumption of DRAM and flash memory for different transfer rates. We can see that as the transfer rate increases, the power requirement for flash memory writes becomes larger than for DRAM. This crossover occurs at a transfer rate of approximately 5KB/s which is marked but not labeled on the figure. We can also see that flash reads always consume less power than DRAM reads or writes. The DRAM self refresh power is 6.6mW and is very close to the active current when the transfer rate is low. The flash memory write will not work at a higher transfer rate than 162KB/s with a power consumption of 198mW. A transfer rate of

162KB/s is marked but not labeled in Figure 1.3. At an average transfer rate of 1MB/s the power consumption of the DRAM is 9.18mW, and flash reads are 1.52mW.

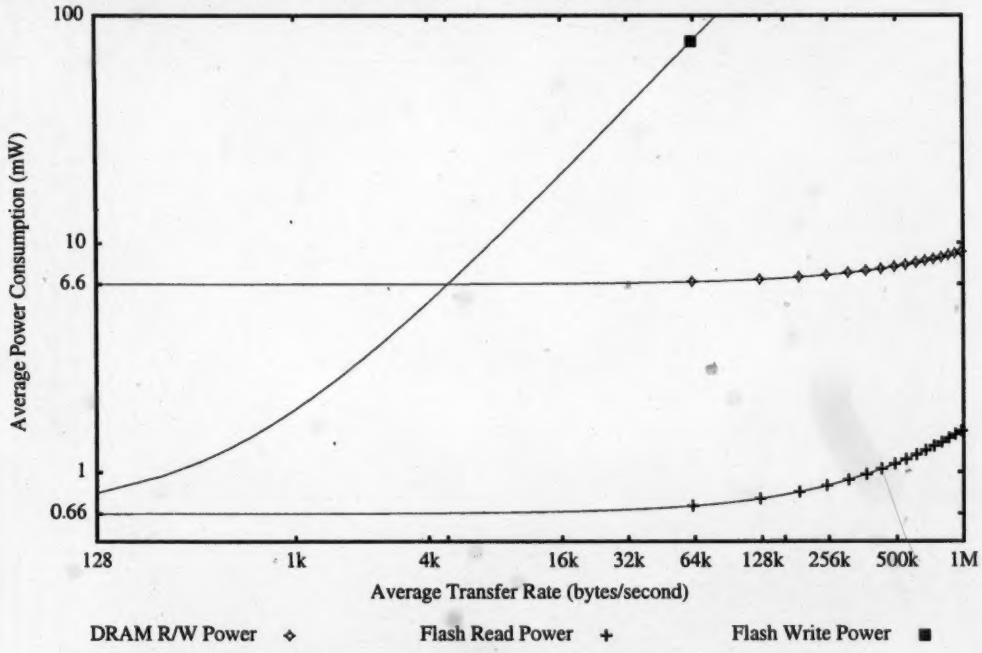


FIGURE 1.3. Power Consumption as a Function of Transfer Rate

Other potential applications such as laptop computer solid state disks and low latency web/media servers are shown below in Table 1-2 with their estimated transfer rates. Note that DRAM would have a power advantage over flash in all such applications if writing were to be a common operation. Even in the case of a mostly-reading MP3 player, DRAM enjoys a significant cost advantage over flash memory.

TABLE 1-2. Power Consumption for Various Applications

Application	Average Transfer Rate
MP3 player	16KB/s
Digital Camera	34KB/s
Laptop solid state disk, diskless computer, set-top box	> 15MB/s
Low latency web/media server	5.6MB/s (Telecom DS3)

1.4 Microprocessor versus DRAM Feature Size

Prototype DRAM chips use some of the finest feature size found in industry and typically required three or more years to go from prototype to mass production. During

this time the feature size does not usually change very much for the same DRAM chip generation. Microprocessors go into mass production soon after being prototyped and use a larger feature size than DRAM chips during prototyping. Once microprocessors go into production they use a smaller feature size than the mass-produced DRAM chips. We can now examine all prototype DRAM chips and microprocessors and compare their feature sizes and see how they change over the years. The feature sizes reported in DRAM papers from the ISSCC from 1990 to 1999 are shown below. The DRAM papers go from [28] to [55] and show which DRAM prototypes were published.

TABLE 1-3. Feature Sizes for Reported DRAMs

Year	1Gb (μm)	256Mb (μm)	64Mb (μm)	16Mb (μm)	Area (mm^2)	Reference
1990				0.5	7.8x18.06	[54]
1990				0.7	5.63x15.2	[55]
1991			0.4		9.22x19.13	[49]
1991			0.4		11.27x19.94	[50]
1991			0.4		12.5x18.7	[51]
1991			0.4		10.85x21.60	[52]
1991			0.3			[53]
1992			0.4		19.48x9.55	[48]
1993		0.25			13.6x24.5	[45]
1993		0.25			14.4x33.2	[46]
1993		0.4			20.0x23.2	[47]
1994		0.25			13.2x25.9	[42]
1994		0.25			16.4x25.1	[43]
1994		0.25			13.32x22.84	[44]
1995	0.25				936	[38]
1995	0.16				19.25x37.12	[39]
1995		0.3			24.8x14.6	[40]
1995			0.25		15.91x9.06	[41]
1996	0.16				31.26x20.86	[35]
1996	0.15				24.19x24.05	[36]
1996		0.25			21.0x11.7	[37]
1998	0.18				17.37x29.07	[32]
1998		0.22			11.50x17.82	[33]
1998		0.22			12x18.3	[34]
1999	0.175				14.3x27.3	[28]
1999	0.18				17.75x30.80	[29]
1999			0.25		20.5x12.1	[30]
1999	0.14				29.0x12.0	[31]

Microprocessor papers from ISSCC from 1990 to 1999 were also gathered for comparison purposes and are listed below from [56] to [83].

TABLE 1-4. Feature Sizes for Reported Microprocessors

Year	Process (Technology) (μm)	Clock Speed (MHz)	Number of Transistors	Area	Reference
1990	l_{eff}=0.9	90	479k	1.4x1.4cm^2	[81]
1990	1.5	50	77k	78mm^2	[82]
1990	0.8	40	1M	14.85x15.13mm^2	[83]
1991	l_{eff}=0.6	100	1.2M	80mm^2	[80]
1992	0.8	40	3.1M	15.98x15.98mm^2	[77]
1992	0.75	200	1.68M	16.8x13.9mm^2	[78]
1992	0.75	100	1.3M	1.62x1.46cm^2	[79]
1993	0.8	40	1.71M	16.3x12.7mm^2	[76]
1995	0.5	300	9.3M	16.5x18.1mm^2	[70]
1995	0.5	66	1M	7.07x7.07mm^2	[71]
1995	0.5	167	5.2M	17.7x17.8mm^2	[72]
1995	0.5	133	6.88M	18.2x17.2mm^2	[73]
1995	0.5	93	3.5M	14.1x14.1mm^2	[74]
1995	0.4	154			[75]
1996	0.35	433	9.6M	14.5x14.4mm^2	[65]
1996	0.6	120	2.4M	11.6x13.8mm^2	[66]
1996	0.5	200	2.68M	7.5x10.5mm^2	[67]
1996	0.35	160	2.1M	7.8x6.4mm^2	[68]
1996	0.5	250	3.8M	17.68x19.1mm^2	[69]
1998	0.25	100		80mm^2	[61]
1998	0.2	480	6.4M	67mm^2	[62]
1998	0.25	450	7.5M	131mm^2	[63]
1998	l_{eff}=0.18	350	121.5M	162mm^2	[64]
1999	0.25	600	9.5M	10.17x12.10mm^2	[56]
1999	0.2	450	10.5M	83mm^2	[57]
1999	l_{eff}=0.16	500	22M	1.8cm^2	[58]
1999	0.25	600	25M	14.7x14.7mm^2	[59]
1999	0.25	500	116M	21.3x22mm^2	[60]

In Figure 1.4 we plot the feature sizes for each year and see how DRAM and microprocessors compare over time. The effective lengths of microprocessors were not plotted because that is not a fair comparison in terms of the technology. We can see that microprocessors use a larger feature size than DRAM chips. As technology is improved, the feature size used by microprocessors becomes smaller than the feature size used in DRAM production. In Figure 1.4 each production year is in bold. The initial mass production years for 4, 16, 64, and 256Mb DRAM chips were 1990, 1993, 1996, and 1999, respectively. This means that in 1993 16Mb DRAM became cheaper to manufacture than 4Mb DRAM. Mass production feature size for each of the DRAM chips are missing from the figure, but we can see for example in 1998 that the feature size of a 256Mb DRAM is larger than the feature size used for a microprocessor. This shows that, while prototype DRAMs use a smaller feature size than prototype microprocessors, when it comes to production, microprocessors use a smaller feature size. We assert that DRAMs with a smaller feature size could economically enter production earlier (perhaps the same time as the microprocessors) if the yield problems were addressed.

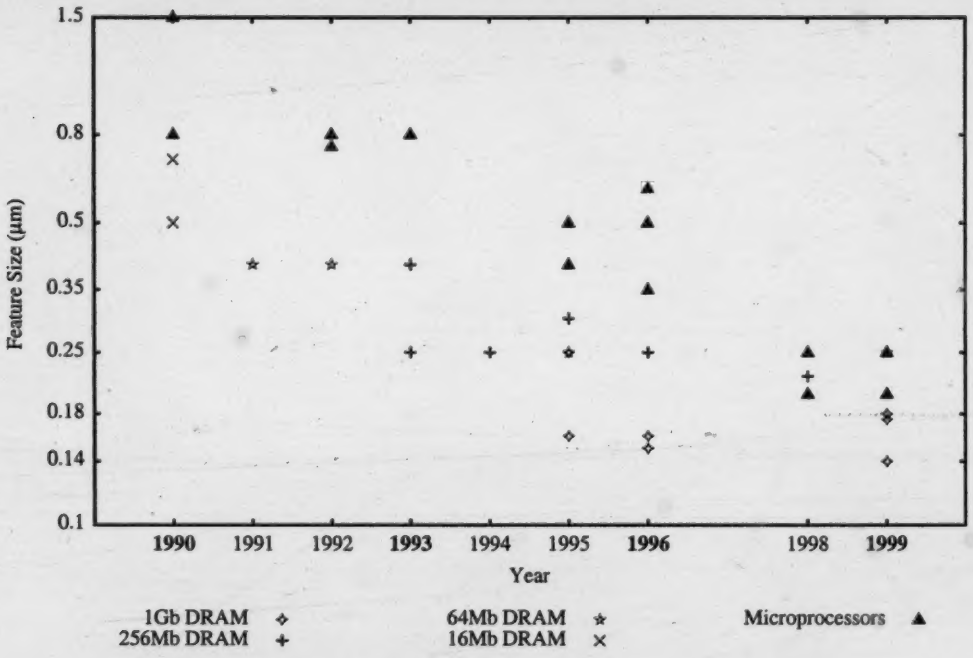


FIGURE 1.4. DRAM and Microprocessor Feature Size Over Time

1.5 Organization of the Thesis

The rest of this thesis is organized as follows: Chapter 2 reviews the theory of yield and cost modeling, and shows how the analysis can be extended to memories with fault tolerance mechanisms. Chapter 2 also reviews error correction codes (ECC) and shows how they affect the yield. Chapter 3 introduces bad block marking which is a method of

marking solid state memory as bad and increasing the yield. Several different schemes are examined for BBM and their effectiveness is compared. Chapter 4 gives the measured software execution time for off-chip ECC correction, and bad block marking software schemes. This permits one to quantify the performance penalty if these schemes are used. Chapter 5 contains an implementation of ECC and determines the area required using place and route tools. An analysis of the expected size of ECC circuitry if it were extended to a more modern 1Gb DRAM is also presented. Chapter 6 concludes the thesis with a summary of the major results and a discussion of future work.

Chapter 2: Effects of Conventional Fault Tolerance on Yield

This chapter investigates and compares several fault tolerance techniques and determines their effect on yield. The techniques that are investigated are: the use of row redundancy, the use of column redundancy, and the use of Error Correction Codes (ECC). When combining redundancy techniques it is possible to achieve *synergistic fault tolerance*, which is also explained. Different redundancy schemes can be compared using a measure called equivalent yield. The way in which the use of Hamming codes effect soft and hard error rates is also examined. Hard errors are manufacturing defects and soft errors are transient errors. Both of these will be explained in greater detail later in this chapter.

2.1 Synergistic Fault Tolerance

The IBM 16Mb ECC DRAM [1] used a Hamming code to both increase the reliability against soft errors and to correct hard errors. Soft errors are transient errors that can occur randomly and occur due to excess noise, retention time being violated, or alpha radiation. They are anything that causes a cell to change value after being successfully tested. Hard errors are errors that occur during manufacturing and always present. Hard errors are manufacturing defects such as bitline shorts, wordline failures, or cell failures. In addition to providing on-chip single error correction double errors detection (SEC-DED) coding and decoding circuitry (based on an ECC similar to a modified Hamming code), the IBM design kept with conventional practice and provided both redundant rows and columns. A key observation was that using redundancy together with error correcting circuitry (ECC) produced a far greater fault tolerance that just a simple sum of the fault tolerance provided by the redundancy and ECC alone. This is illustrated below in Figure 2.1. Using redundancy alone, a 50% expected yield can be achieved in the presence of an average of 28 failing cells. Using ECC alone, 50% yield could be achieved in the presence of 428 failing cells. However, with both redundancy and ECC, an average of 2725 failing cells could be tolerated to achieve a 50% yield. These yield points are taken from [2]. The synergism between redundancy and ECC is due to the complementary abilities of the two fault tolerance methods. Redundancy can be used to "break up" clusters of faulty bits so that it is rare that an ECC codeword has more than 1 faulty bit. The ECC can then readily detect and correct the resulting isolated faulty bits. The synergistic effect was so strong that, in the opinion of the IBM authors, the fault-tolerance of their DRAM greatly exceeded the foreseeable fault tolerance requirements of any fabrication process at that time (the early 1990's). We, however, are trying to define methods that can be used in order to bring forward the economic cross-over point between current mass manufactured 64Mb and 256Mb DRAM chips and larger, next generation (1Gb and 4Gb) DRAM chips. The relaxed constraints of file store memory may benefit from synergistic fault tolerance.

The IBM chip used ECC codewords containing 137 bits. Each codeword contained 128 data bits and 9 check bits. The codeword length appears to have been selected mainly to produce acceptable random-access times. The extra area required to store the check bits

and the extra area occupied by the ECC circuitry increased the size of the ECC DRAM by about 11% [2] over the size of a conventional DRAM with only row and column redundancy [1].

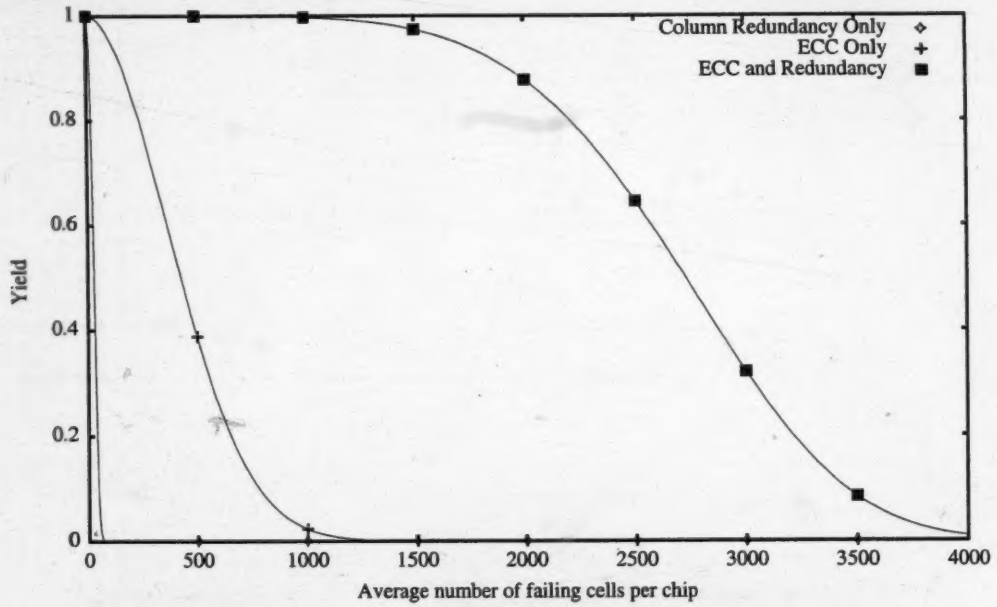


FIGURE 2.1. IBM Redundancy, ECC, and Redundancy/ECC plots [1]

The majority of our work presented here (with the exception of Section 2.4.5) was done using the relatively well documented IBM ECC DRAM as a reference point.

2.2 Yield

Yield is defined as the percentage of dies that get manufactured that are functional. It is between 0% and 100%. The yield changes depending on the die size, and the defect density. It also changes depending on the redundancy on the die. It will be shown in Section 2.7 how the yield is calculated. Using the yield to compare die with redundancy is not acceptable because it does not account for the area required by the redundancy compared to no redundancy. We will use a term called equivalent yield which is defined below. The model that will be used for the yield will be a Poisson model. We are using a Poisson model because it is mathematically simple and was also used in [2]. It was shown in [2] that when a more complex clustered-defect model was used that the synergistic fault tolerance effect was still achieved. The Poisson distribution is used when calculating the cell yield based on the average number of defects.

2.3 Modeling Using Equivalent Yield

In Section 2.1, plots of conventional yield were used to compare redundancy methods. However, using yield to measure the effectiveness of a redundancy method misses the key economic goal of reducing the average cost per working bit. To take area into account, the ratio of yield and area will be used to compare redundancy techniques. This yield to area ratio is a normalized yield that better captures the economic goal of reducing the cost per working bit. An equivalent yield can represent a partially good product and is described in [22]. The area is roughly proportional to the total number of bits on a die and can thus be replaced with the total bits per die. Multiplying this ratio by the number of working data bits on the die produces the formula that we will use for the equivalent yield, shown in Equation (2.1). Note that the equivalent yield is equal to the reciprocal of the cost per working bit (cpwb).

$$Yield_{Equivalent} = \frac{Yield \times UsableBitsPerDie}{TotalBitsPerDie} = \frac{1}{cpwb} \tag{2.1}$$

where *cpwb* stands for the cost per working bit. The yield was calculated using a Poisson model as described in [1]. This model was used to calculate the yields for this work, for example, the curves in Figure 2.1. Following the IBM work, we are considering single cell failures that are uncorrelated in position.

2.4 Hamming Codes in DRAM

One possible method for increasing the yield of a DRAM chip is the use of an error correcting code (ECC). There are several advantages and trade-offs when using an ECC. The many possible design choices include: whether to use a Modified Hamming code versus a normal Hamming code; the ECC word size; and whether or not both detection and correction, or only detection, should be performed.

2.4.1. Hamming Codes

Hamming codes are an efficient class of error-correcting codes that have been implemented both in memory modules and in ECC DRAMs [2][26]. Check bits are calculated from data bits and are stored with the data bits in memory. When the data is read back from memory, the check bits are recalculated and compared to the bits that were written originally. This comparison creates syndrome bits which indicate whether an error has occurred. If the Hamming codes are chosen carefully and have sufficient and appropriately defined check bits then it becomes possible to determine if two errors have occurred. This type of error correction code, which can correct one error and detect two errors, is called a single error correcting, double error detecting (SEC-DED) ECC code. A Hamming code can also be chosen such that it is a SEC-DED code. A Modified Hamming code is by definition a SEC-DED code. Modified Hamming codes will be described later in Section 2.4.3.

2.4.2. Hamming Code Operation

Each codeword of length n consists of k data bits augmented with r check bits (so that $n=k+r$). The relationship between parameters k and r is given by Equation (2.2)

$$k \leq 2^r - r - 1 \tag{2.2}$$

First there are a total of 2^r possible combinations for codes, we must now subtract r from the number of codewords because the single weighted codes are used for the check bits, and we must subtract one because the all zero case cannot be a matrix combination because it indicates that no errors have occurred. A Hamming code encodes the k data bits into the r check bits which are then stored in memory with the data. When the data is read back from memory, the check bits are recalculated from the data and then compared to the check bits that were originally written to memory. This comparison produces r syndrome bits which reveal whether a single error has occurred.

Hamming code Matrix

Hamming codes are specified by the total number of bits in the code and by the number of data bits. Table 2-1 below shows a simple (13,8) Hamming code (13 bits total, with 8 data bits). The matrix given below in Table 2-1 is actually a Modified Hamming code, which will be described in greater detail in Section 2.4.3. Every '●' in the matrix corresponds to one input to an XOR gate tree. Check bits are calculated as the XOR of all the ●'s in one row in the "Data Bit" section. The syndrome bits are then created by taking the bitwise XOR of the check bits read from memory and the recalculated check bits (i.e. all the ●'s in each row under both the Data and Check Bit sections). The syndrome bits indicate if an error has occurred and, depending on the code, they may also indicate the location of the error. An all-zero syndrome vector indicates that no errors were detected.

TABLE 2-1. Hamming Code Matrix

(13,8) Modified Hamming Code Matrix													
Data Bit								Check Bit					
7	6	5	4	3	2	1	0	5	4	3	2	1	Syndrome Bit
		●	●		●	●	●					●	1
	●		●	●		●	●				●		2
●		●		●	●		●			●			3
●	●			●	●	●			●				4
●	●	●	●					●					5

Several different larger Hamming codes are given in Appendix B.

2.4.3. Modified Hamming Codes

Modified Hamming codes are a specific sub-class of Hamming codes. Every Modified Hamming code is thus a Hamming code, but every Hamming code is not necessarily a Modified Hamming code. In order for a Hamming code to be considered a Modified Hamming code it must satisfy the following three criteria.

The first criterion is that an odd-weight code of 3 or greater must be used. This constraint reduces the number of possible codewords by 2. An odd weight code of 3 or greater means that the number of (●'s) in a column is an odd number larger than three. The second criterion is that the number of gate inputs (●'s) in the Hamming code matrix must be minimal. This constraint tends to reduce the required hardware and speed up the calculation of the syndrome bits. The last constraint is that the number of inputs for each of the syndrome bit gates must be as balanced in weight as possible.

Using a Modified Hamming code will minimize the number of transistors needed to create the error detection and correction hardware. Using a Modified Hamming code, however, does not guarantee a minimal ECC hardware area. A Modified Hamming code does not take into account the amount of area required for wiring. The IBM DRAM, for example, did not use a Modified Hamming code but used Hamming codes that allowed a more regular layout. Their code did, however, satisfy the odd-weight code criteria and it guaranteed single error correction and double error detection.

Odd-Weight Codes

If an odd-weight code is used it becomes possible to correct a single error and to detect whether a double error has occurred [25]. The maximum number k of protected data bits given r check bits (for a SEC-DED code) is given below in Equation (2.3).

$$k \leq 2^{r-1} - r \tag{2.3}$$

In Equation (2.3), r is the number of check bits, and k is the number of data bits that will be protected. We first calculate the number of odd weight codes by subtracting one from r and putting it to the power of two (2^{r-1}). We then subtract r from the number so that the codes with a weight of one can be used for the check bits. We do not need to subtract 1 from Equation (2.3) as we did in Equation (2.2) because the all zero case is considered even and is removed with the r-1. As can be seen from Equation (2.3), there is a limit to the number of data bits that can be protected for any given number of check bits. Table 2-2

shows the maximum number of data bits that can be protected against errors and typical configurations using *r* check bits with a Modified Hamming code.

TABLE 2-2. Modified Hamming Code Check Bits

Number of Check Bits (r)	Maximum Number of Data Bits ($k=2^{r-1}-r$)	Typical Number of Data Bits (k)
5	11	8
6	26	16
7	57	32
8	129	64
9	247	128
10	502	256
11	1013	512
12	2036	1024
13	4083	2048
14	8178	4096

Minimizing Gate Inputs in the Hamming Code Matrix

The number of data bits that need to be protected is typically less than the maximum number of data bits that can be protected. This means that we must choose which Hamming codewords we want to use. For example, for a (39,32) code there are 57 possible Hamming code matrix rows. Of these 57 possible Hamming codewords there are 35 with a weight of 3 (calculated as 7 choose 3), 21 with a weight of 5 (calculated as 7 choose 5), and one with a weight of 7 (calculated as 7 choose 7). We must now choose 32 of these possible rows to generate the code matrix. In order to minimize the number of inputs in the matrix, the lowest weighted row must be selected first.

Balanced Inputs for Syndrome Bits

The last criteria is that the number of inputs for each of the syndrome bits should be balanced. This corresponds to selecting code matrix rows that balance out the inputs as much as possible. The code illustrated in Table 2-1 has balanced inputs of 5, 5, 5, 5, and 4.

Correcting Data

If an error has occurred then the r-bit-long nonzero syndrome bit vector can be decoded and an n-bit-long bit mask can be created. The mask is then used to correct the data bits by bitwise XORing the mask and the data. Descriptions of how this is done for the Modified Hamming code are given in [25] and [26].

2.4.4. Hamming Codeword Size

If a Modified Hamming code is used in a DRAM, then an architectural design issue is how long should the ECC codeword be? A 137-bit ECC codeword with 128 data bits was used in the IBM ECC DRAM. In [12] it was stated that "comparisons of 72-, 137-, and 266-bit ECC systems show that maximum density is at the 137-bit word length and the change in speed from 72-bit system to a 137-bit system is a fraction of a nanosecond." The definition of "maximum density" was not clearly explained. It appears likely that they were referring to the maximum layout density of the Hamming code ECC circuitry.

We modeled the yield of a chip using the equations from [22]. To revisit the choice of codeword length, the codeword length was varied, then the resulting equivalent yield was plotted. This was done in order to see what effect changing the codeword length would have on the equivalent yield. The equations required to plot this figure are not presented here but will be presented later in Section 2.8, "Effects of ECC, Redundancy, and ECC & Redundancy on Yield". The specific equations needed are Equation (2.7), Equation (2.19), and Equation (2.20) with the number of ECC bits being varied in Equation (2.19). The equivalent yield is shown in Figure 2.2.

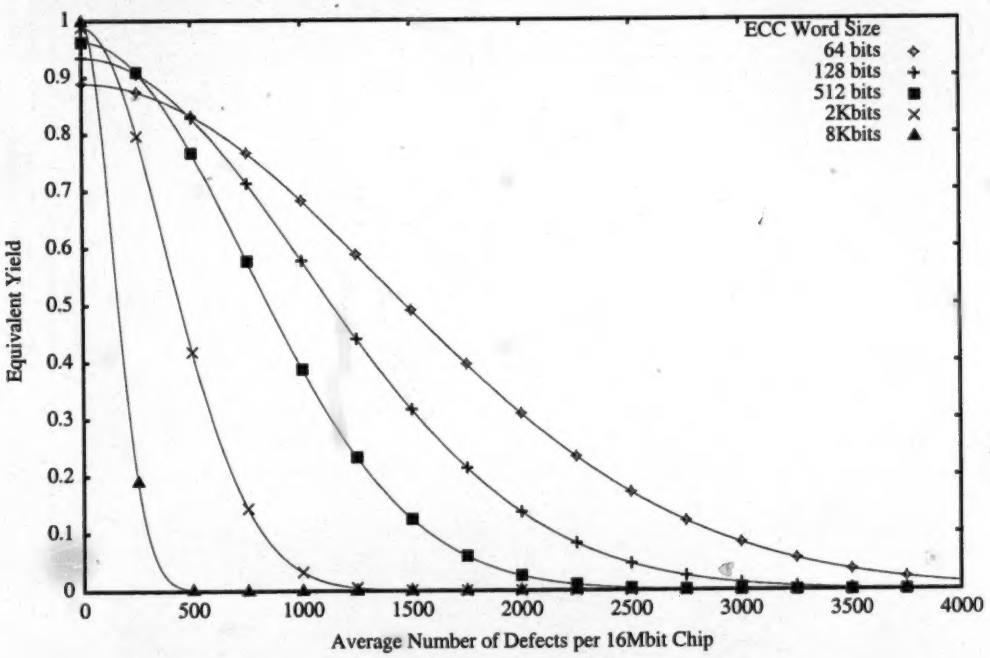


FIGURE 2.2. Effects of Different Codeword Sizes on the Equivalent Yield

While the *conventional yield* is 100% when there are no defects, the *equivalent yield* is less due to the area overhead of the ECC bits. Remember that the equivalent yield accounts for the area requirement for the redundancy technique being used. Figure 2.2 shows that as the ECC codeword length increases, the equivalent yield increases for low

numbers of defects. This is because, as the length is increased, the fraction of bits needed to hold the ECC bits decreases overall and thus gives us a higher equivalent yield as well as a greater ratio of data bits to total bits. Although lengthening the codewords increases the equivalent yield when the average number of faults is low, it also means that the dies cannot tolerate more than a very small number of defects. Overall, shorter codewords produce a more graceful degradation in yield as the expected number of failing cells increases. For any average number of faults there is always an optimal length for the ECC word. There are three practical limitations on how long the ECC word can be made. The first is that all of the codeword bits need to be read in order for an error to be detected and corrected. This implies that the upper limit on the size of the ECC word may be dictated by the size of a DRAM page, which is the size of data that can be brought into the sense amps at a time inside a DRAM. The second limitation is that as the ECC word becomes longer, the area of the circuitry to detect and correct the error increases. Finally, the error detection and correction processes are increasingly slower for larger codewords.

2.4.5. Area of Hamming Code Circuitry

2.4.5.1 Hamming Code Hardware

Our analysis has not yet taken into account the amount of area needed to perform detection and correction. Since processes and feature sizes are always changing, the area needed by the ECC circuitry can be estimated in terms of the number of 2-input, 1-output, 4-transistor CMOS gates. Any *n*-input gate can be built from 2-input gates by using a tree structure. Since ECC circuitry typically contains many XOR gates, one could argue that a 2-input, 1-output XOR gate cannot be built with 4 transistors. However, fast 4-input XOR gates can be built in 16 transistors using a DCVS XOR gate as described in [23]. This comes close to the assumed 12 transistors and is close enough for a first-order approximation.

Hardware Circuitry

Since there are many different ways in which the hardware could be implemented, we chose a straightforward hardware architecture from [26], as shown below in Figure 2.3 and Figure 2.4.

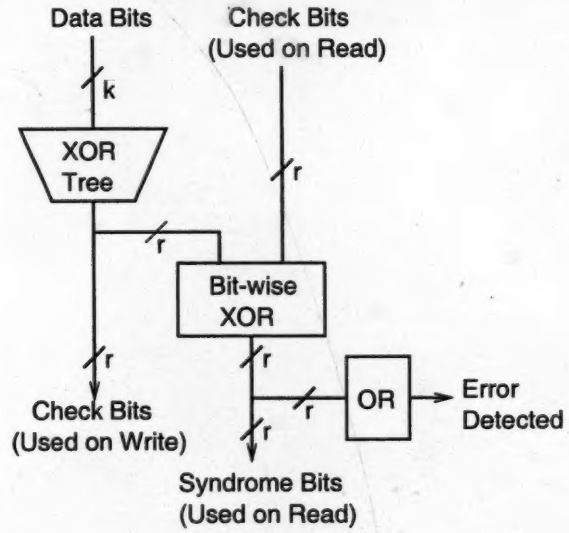


FIGURE 2.3. Hamming Code Error Detection Hardware [26]

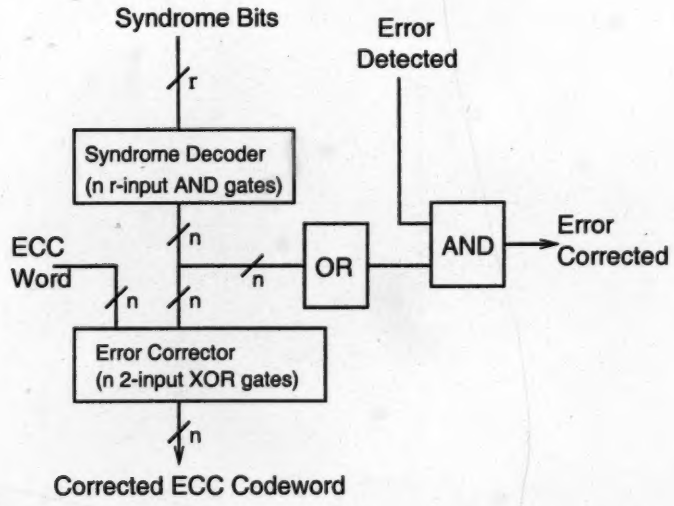


FIGURE 2.4. Hamming Code Error Correction Hardware [26]

Figure 2.3 shows the gates required in order to detect that an error has occurred, and Figure 2.4 shows the gates required to correct a detected error. After the syndrome bits have been created, they are decoded and a mask is created. This can be accomplished by using n r-input AND gates, where n is $k+r$. The mask is then used to flip incorrect bits, and thereby correct the codeword bits, by using n two-way XOR gates. There is some additional circuitry that is used to determine whether an error has occurred and whether the error has been corrected. This circuitry is shown in Figure 2.4.

Normalized Area

Figure 2.5 below shows how the normalized area of the above Modified Hamming code circuitry grows with codeword length. As the number of check bits is varied from 8 to 10 bits we can see that, in all cases, the area grows in a roughly linear fashion with the number of data bits in the codeword. We can also see that the area needed for detection and correction is significantly larger than the area required for detection only. This is an important factor to note because, if we are not concerned with performance, then we could consider off-loading error correction (an infrequent operation) to off-chip logic, a microprocessor, or the host processor. This would reduce the cost of ECC further. This could in turn allow the ECC word to be made larger while reducing the ECC hardware. The penalty would be a slower average access time, depending on the access patterns and the location of the errors.

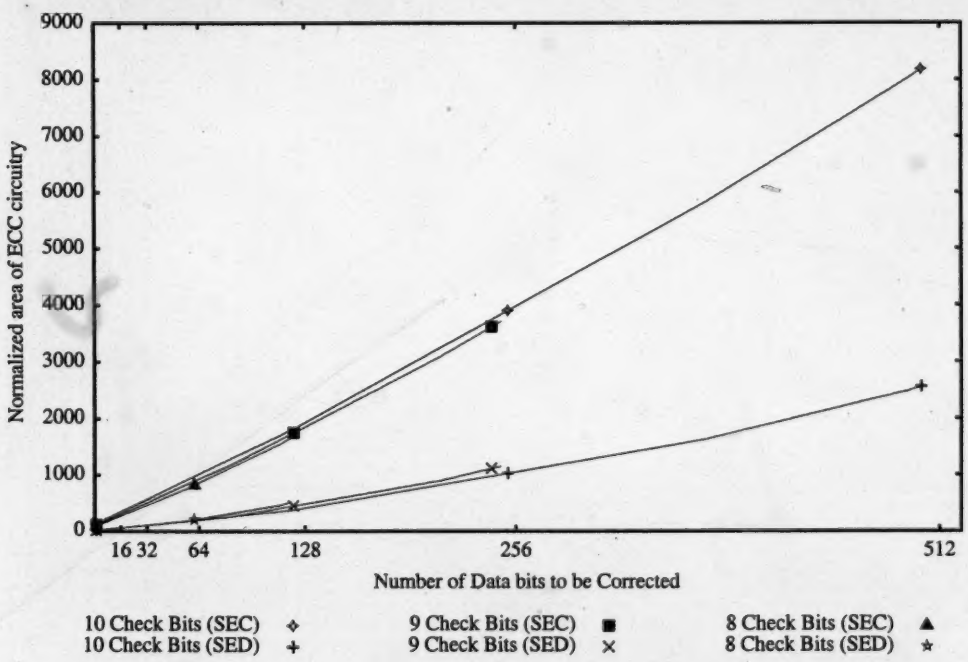


FIGURE 2.5. Comparison of ECC Modified Hamming Code Circuitry Area

Table 2-3 shows the normalized area needed for doing detection and correction, the normalized area needed for detection only, and the area ratio between the two. The first three rows show that as the number *k* of data bits increases, the area ratio decreases. The last four rows show what effect changing the number of check bits has on the area ratio. Since we would typically increase the number of check bits in order to accommodate longer codewords, this comparison was done with each check bit size having the largest power of 2 data bits that it can correct. We can see that this ratio varies within the relatively tight range of 3.8-4.0. This implies that a significant area (about 75%) can be saved if the syndrome bits are sent off-chip to perform the ECC calculation.

TABLE 2-3. Area of SEC-DED vs. DED

		Area		
r	k	SEC-DED	DED	Ratio
9	64	936	206	4.54
9	128	1855	485	3.82
9	247	3711	1151	3.22
8	64	863	215	4.01
9	128	1855	485	3.82
10	256	3975	1049	3.79
11	512	8519	2243	3.80

2.4.5.2 Hamming Code Delay

Figure 2.6 shows the estimated delay (in terms of gate delays) of the ECC circuitry as a function of the number of data bits to be corrected for SEC-DED. It is fairly obvious from the figure that the delay increases roughly logarithmically with the number of data bits. This is simply because, as the number of data bits increases, the depth of the XOR tree needed to calculate the check bits increases in depth logarithmically. The flat plateaus seen

are due to the fact that the depth of the tree does not always increase when more bits are added.

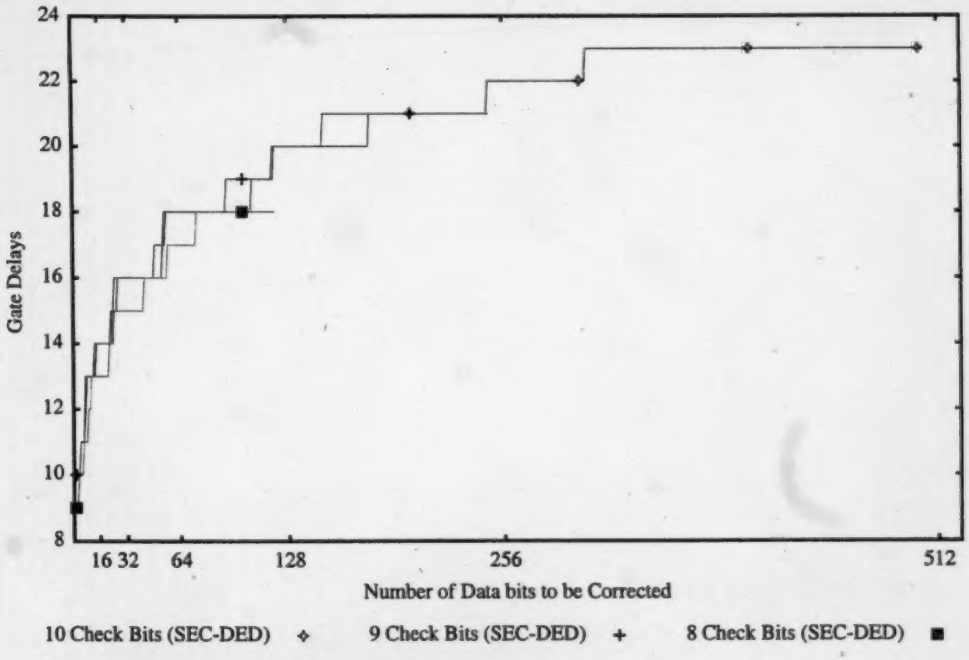


FIGURE 2.6. SEC-DED ECC Circuit Delay

Figure 2.7 below shows the performance difference, in terms of gate delays, between doing SEC-DED and doing only DED. The plotted circuit delay corresponds to the number of gate delays for the error detected signal to be generated. We can see from the figure that the DED takes approximately half the number of gate delays. As the number of

bits to be corrected increases, the ratio of DED delay to SEC-DED delay decreases slowly from about 1.2 at one data bit to 1.08 at 500 data bits.

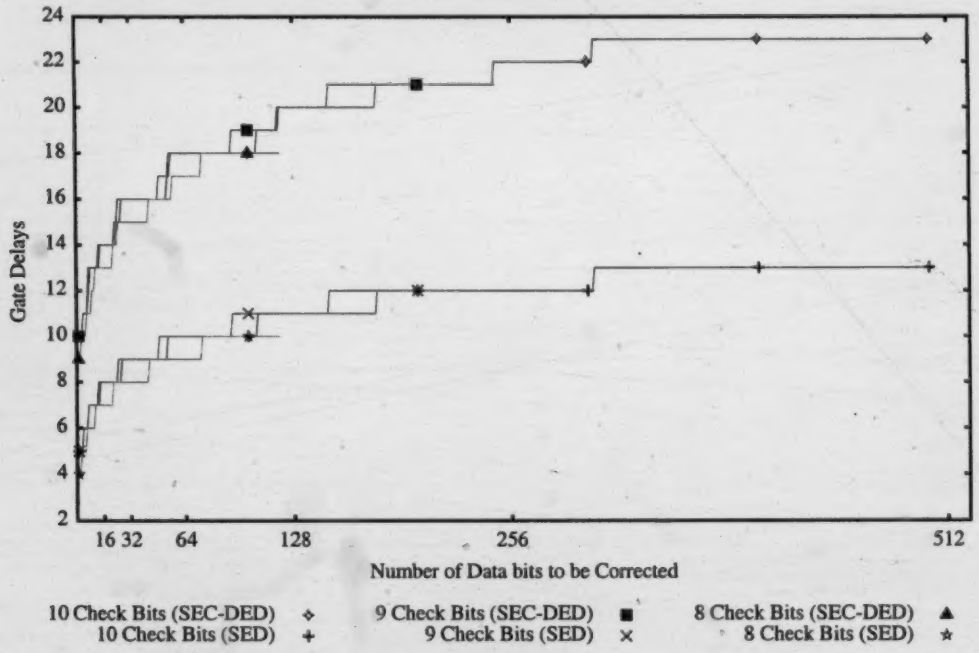


FIGURE 2.7. SED ECC Circuit Delay

2.5 Using ECC to Improve Both Hard and Soft Error Rates

Soft errors in DRAM are flipped bits caused by internal or environmental factors. Soft error rates are measured as the number of failures in time (FIT). FIT is defined as the number of errors that occur in 10^9 hours or the mean time between failures is 10^7 hours. For example 100FIT means that there are an average of 100 errors in 10^9 hours. ECC can be used to correct hard errors and thus improve the yield of the manufacturing process, as shown in the previous sections. ECC can, however, also be used to decrease the soft error rate of DRAM. Typical DRAM chips are 100FIT [84]. This means that ECC could be used to decrease the mean time between failures (MTBF) of a DRAM that otherwise could not be sold because the MTBF is too high. This could allow smaller cells or multilevel DRAM (MLDRAM) [4][5][7] that are more susceptible to soft errors to be used. Soft errors can be corrected in an ECC DRAM by reading each codeword and thereby triggering error detection and correction. This process is called scrubbing and *ScrubInterval* is the time between scrubbing. We can take what the FIT of a DRAM that does not use ECC and

calculate what the FIT of the DRAM would be if it used ECC. We can do this using Equation (2.4), Equation (2.5), and Equation (2.6)

$$P_{se} = \frac{FIT}{DRAMSIZE} ScrubInterval \tag{2.4}$$

where P_{se} stands for the probability of a soft error occurring per cell, $ScrubInterval$ is the amount of time between subsequent reads to the same codeword, and $DRAMSIZE$ is the number of bits in the DRAM.

$$P_{seecc} = 1 - ((1 - P_{se})^{ECCbits} + ECCbits \cdot (1 - P_{se})^{(ECCBits - 1)} \cdot P_{se}) \tag{2.5}$$

Where P_{seecc} is the probability of having a soft error in a ECC word that cannot be corrected, and $ECCbits$ is the number of bits in an ECC word. An uncorrectable soft error corresponds to having two or more errors in the ECC word within one $ScrubInterval$. The FIT of the DRAM using ECC can then be calculated by multiplying P_{seecc} by the number of ECC words ($ECCWords$) in the DRAM and dividing by $ScrubInterval$.

$$FIT_{ECC} = \frac{P_{seecc} \cdot ECCWords}{ScrubInterval} \tag{2.6}$$

It should be noted that the FIT and $ScrubInterval$ must be in the same units. This means that if $ScrubInterval$ is measured in seconds then the FIT must be converted to the number of errors per second. Figure 2.8 shows how the FIT of a DRAM changes when ECC is used. The $ScrubInterval$ was set to 1000 seconds for this figure. We can see from the figure that there is a very large decrease in the FIT when ECC is used. We can also see that the FIT depends on the page size. There are three different lines in the plot. Two of

them represent a 1Gb DRAM with different codeword lengths, and the last is how the IBM 16Mb DRAM was effected by the use of ECC.

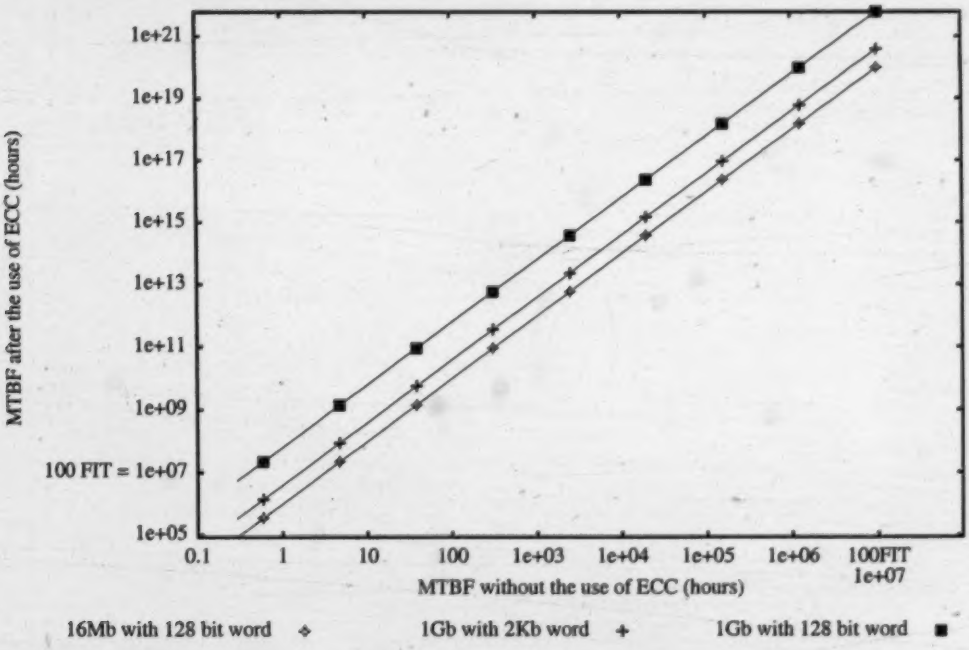


FIGURE 2.8. How ECC Affects FIT for a 16Mb and a 1Gb DRAM

2.5.1. Hard Error Correcting Affecting Soft Error Rate

It has been shown above that using ECC results in higher reliability. This assumed, however, that all of the ECC's fault tolerance was used to correct soft errors. It was shown in earlier sections how the use of ECC can result in higher yield by using ECC to correct hard errors. If the ECC is used to correct hard errors, and thus increase yield, then the ECC can no longer be used to correct a soft error within the same codeword. Figure 2.9 below shows how the mean time between failure (MTBF) is effected by the average number of defects for the 16Mb DRAM. As the average number of random hard defects increases, the number of ECC words that need to be used to correct hard errors increases and the

effectiveness of the ECC at correcting random soft errors decreases. This can be seen in Figure 2.9.

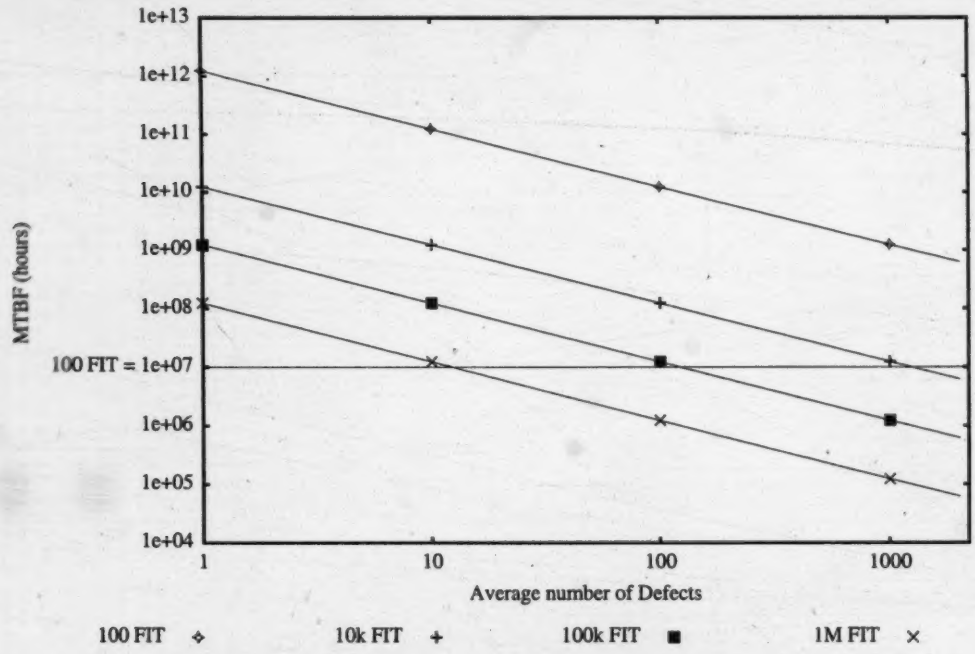


FIGURE 2.9. Acceptable Soft Error Rate Using ECC using 16Mb

Four different lines are plotted in the figure. The first is for a 100FIT DRAM that uses ECC. We can see that, for all defect levels, the DRAM has a lower soft error rate than if ECC had not been used. Since the DRAM always performs better, this means that we can relax the constraint that the DRAM meet 100FIT or better before ECC. The other three lines in the graph show how DRAM chips with 10k FIT, 100k FIT, and 1M FIT respond to an increasing average number of defects. If the defect level is low, then all of these cases perform better than a standard 100 FIT DRAM. However, if we increase the number of errors, we can find the points where the DRAM chip becomes as reliable with respect to soft errors as a 100 FIT DRAM. A 10k FIT, 100k FIT, and 1M FIT are equivalent to a 100 FIT DRAM that does not use ECC at 1320, 130, and 10, defects respectively.

It should be noted that, in the presence of an average of 1000 defects, the yield is approximately 2% and it is approximately zero percent at 1180 defects. There numbers can be found in Figure 2.11 for the equivalent yield for ECC only in Section 2.9.This means that even though ECC can be used to correct soft errors, there would be very few working chips because the ECC could not correct all of the hard errors and there are no functional chips. The effectiveness of ECC against soft errors is not affected significantly by the number of hard errors that are corrected. This is because the yield goes to zero before there are a significant amount of ECC codewords that cannot correct soft errors, therefore, the limitation for correcting soft errors using ECC is the achievable yield with

the use of ECC and not the ratio of the number of hard errors to soft errors. Figure 2.10 shows the reliability of a 1Gb DRAM using an ECC word size of 512 bits. The yield using only ECC is approximately 0.2% at an average of 5000 defects.

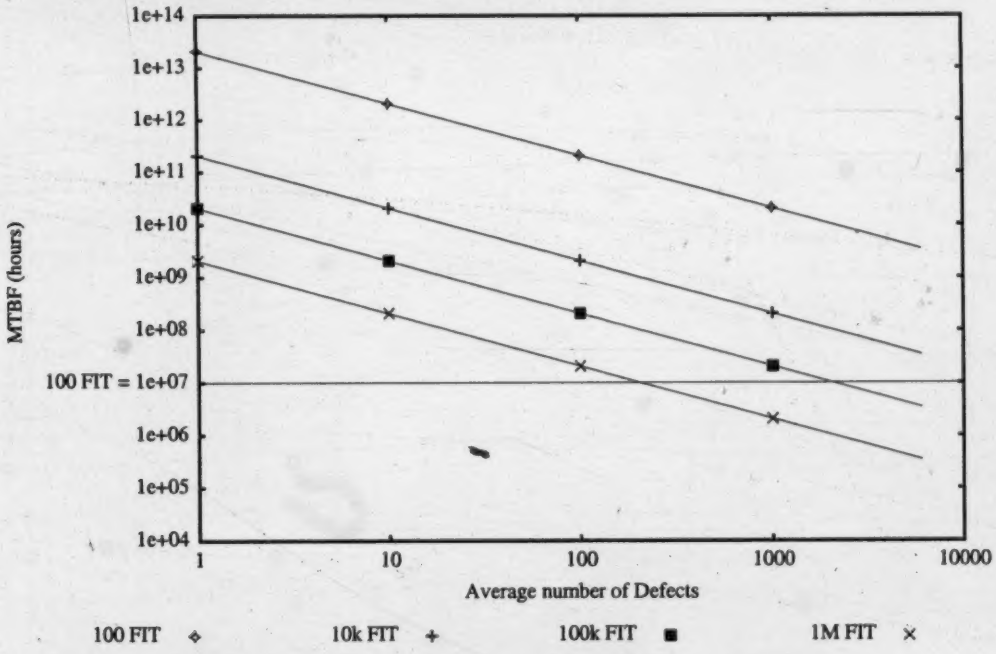


FIGURE 2.10. Acceptable Soft Error Rate Using ECC using 1Gb

2.6 Additional Error Correcting Codes

There exist other more complicated ECC codes, such as Reed-Solomon codes, which can be used to correct errors. They will not be investigated as part of this work and are considered future research. Because of the block-oriented accesses and the relaxed latency requirements, file memories could potentially exploit more powerful coding techniques, such as Reed-Solomon codes, whose check and/or correct cycles were too slow to be considered previously.

2.7 Redundant Rows and Columns

Row and column redundancy techniques will not be looked at extensively because the benefits and implementations of redundancy are already well known. We will, however, examine the effect redundancy has on the yield. Redundancy is already provided in currently manufactured DRAM in order to increase the yield to economical levels. There are several simplifying assumptions that are made. They are that manufacturing defects

only affect one cell (that is to say that we only investigate single cell failures), that the errors are uncorrelated, and that there are no errors in the periphery. These are all optimistic assumptions but are made in order to examine how the equivalent yield changes with the average number of defects. H.L. Kalter made these same assumptions in [2] and also showed how the yield changed when switching to a cluster fault model. The conclusion from using the cluster fault model was that the fault synergistic tolerance effect was not affected by using a different fault model.

2.7.1. The Cost of Redundancy

Since we are now examining the equivalent yield, we need to attribute a cost to the redundant columns and rows. Since the redundant elements are DRAM cells the area required for the redundancy is set to one. This means that if a redundant column is 256 bits then an increase in area of 256 bits is needed. It is assumed that the area required to implement the bit steering (circuitry required for the redundancy to provide the word to the ECC circuit) was 2% when both redundancy, and ECC is used. Since the amount of redundancy is always fixed, the equivalent yield can easily be calculated.

2.8 Effects of ECC, Redundancy, and ECC & Redundancy on Yield

This section describes the equations used to calculate yield when ECC and redundancy are both present. There are a few terms that will be used when describing the use of redundancy in DRAM. A *book* is the block of memory that column redundancy is applied to. A *section* is the block of memory that row redundancy is applied to. In DRAM manufacturing, a section is usually larger than a book. The third term is *ECC codeword*, which is the number of bits that make up a complete ECC word (including data and check bits). The fourth term is the *pageSize*, which is the width of a book in bits.

2.8.1. Yield Assuming No ECC or Redundancy

The Poisson yield model estimates the yield of a die by first considering the yield of a single DRAM cell. The yield of a single DRAM cell can be calculated by using Equation (2.7)

$$Y_{sc} = e^{-\lambda_{sc}} \tag{2.7}$$

where Y_{sc} is the yield of a single DRAM bit and λ_{sc} represents the average number of faults per bit. The average number of faults per bit can be calculated by taking the number of average faults per die and dividing it by the number of bits in the die. The yield of a single DRAM bit is the probability that any given cell will be functional. The average number of faults per cell is directly proportional to the number of defects present in the

array and inversely proportional to the number of bits in the DRAM array. The yield Y_{DRAM} of the DRAM without ECC can then be calculated using Equation (2.8)

$$Y_{DRAM} = Y_{sc}^{BitsInDRAM} \tag{2.8}$$

where $BitsInDRAM$ is the total number of bits in the DRAM.

2.8.2. Yield Using Only Redundant Columns

The yield of a *book* can then be calculated by first calculating the yield Y_c of a column of memory using Equation (2.9).

$$Y_c = (Y_{sc})^{SizeOfRedundantColumn} \tag{2.9}$$

where *SizeOfRedundantColumn* is the number of bits in a column. Next we can calculate the yield of a *book* using Equation (2.10). Equation (2.10) sums over all contributions to the binomial distribution of the yield from zero errors to the number of redundant columns.

$$Y_{book} = \sum_{j=0}^{n_{rc}} \binom{p + n_{rc}}{j} \cdot Y_c^{\,p+n_{rc}-j} \cdot (1 - Y_c)^j \tag{2.10}$$

where n_{rc} is the number of redundant columns, p is the page size (or the number places that the redundant column can be used). It is easiest to understand Equation (2.10) if an example is given. The IBM DRAM has 2 redundant columns and 128 data columns in a book. This corresponds to $n_{rc} = 2$ and $p = 128$. Assuming that the yield of the redundant column is the same as the yield for a regular column of bits then we must have 128 out of the total 130 columns functional in order to have a working book. It does not matter which columns are functional so long as the total is greater or equal to 128. The sum will go from zero column errors (j=0) to two column errors (j=2). The first term in the sum (j=0) will be when all 130 columns are good. The second term (j=1) is 129 good and one bad. The last term (j=2) is 128 good and two bad columns. The first term in the equation is choosing which column is bad, the second term is the yield of all good columns, and the last term is the yield of the bad columns. Now we can calculate the yield of the entire die by using Equation (2.11).

$$Y_{DRAM} = Y_{book}^{NumberOfBooks} \tag{2.11}$$

In Equation (2.11), *NumberOfBooks* is the number of books on the DRAM die.

2.8.3. Yield Using Only Redundant Rows

The yield of a row can be calculated using Equation (2.12).

$$Y_r = (Y_{sc})^{SizeOfRedundantRow} \tag{2.12}$$

In Equation (2.12), Y_r stands for the yield of a redundant row, and *SizeOfRedundantRow* is the number of data bits in a row. The yield of a section can then be calculated using Equation (2.13).

$$Y_{section} = \sum_{j=0}^{n_{rr}} \binom{s+n_{rr}}{j} \cdot Y_r^{\,s+n_{rr}-j} \cdot (1-Y_r)^j \tag{2.13}$$

Where n_{rr} is the number of redundant rows, and s is the section height (or number of places where the redundant row can be used). We can then calculate the yield of the entire die by using Equation (2.14).

$$Y_{DRAM} = Y_{section}^{\,NumberOfSections} \tag{2.14}$$

Where *NumberOfSections* is the number of sections on the DRAM die.

2.8.4. Yield Using Redundant Columns and Redundant Rows

We know that the yield of a book that uses only redundancy can be calculated using Equation (2.10). The yield of the book must be equivalent to the probability of a single cell being good to the power of the number of bits in a book as shown below in Equation (2.15).

$$Y_{sccr}^{\,bitsPerBook} = Y_{book} \tag{2.15}$$

In Equation (2.15), Y_{sccr} is an estimate for the effective yield of a single cell when column redundancy is being used. The yield of a row can then be calculated using Equation (2.16).

$$Y_r = (Y_{sccr})^{SizeOfRedundantRow} \tag{2.16}$$

In Equation (2.16), Y_r stands for the yield of a redundant row, and *SizeOfRedundantRow* is the number of data bits in a row. The yield of a section can then be calculated using Equation (2.17).

$$Y_{section} = \sum_{j=0}^{n_{rr}} \binom{s+n_{rr}}{j} \cdot Y_r^{\,s+n_{rr}-j} \cdot (1-Y_r)^j \tag{2.17}$$

Where n_{rr} is the number of redundant rows, and s is the section height (or number of places where the redundant row can be used). We can then calculate the yield of the entire die by using Equation (2.18).

$$Y_{DRAM} = Y_{section}^{NumberOfSections} \tag{2.18}$$

Where *NumberOfSections* is the number of sections on the DRAM die.

2.8.5. Yield Using Only ECC

The yield of a word that uses SEC ECC can be calculated using Equation (2.19).

$$Y_{eccw} = Y_{sc}^{ECCbits} + \binom{ECCbits}{1} \cdot Y_{sc}^{(ECCbits-1)} \cdot (1 - Y_{sc}) \tag{2.19}$$

In Equation (2.19), *ECCbits* is the number of bits in an ECC word, and where Y_{eccw} stands for the yield of a ECC word or the probability that the ECC word will have zero errors or one (correctable) error. The yield of the book can then be calculated using Equation (2.20)

$$Y_{book} = Y_{eccw}^{ECCwordsPerBook} \tag{2.20}$$

In Equation (2.20), *ECCwordsPerBook* is the number of ECC codewords in a book. The yield of the chip can then be calculated using Equation (2.11).

2.8.6. Yield Using Redundant Columns and ECC

Equation (2.19) above gives the yield of an ECC codeword. We know that the yield of this ECC codeword must be equal to the probability of all the *ECCbits* being good, as shown in Equation (2.21).

$$Y_{scecc}^{ECCbits} = Y_{eccw} \tag{2.21}$$

In Equation (2.21), Y_{scecc} stands for the effective yield of a single cell when ECC is being used. We can then calculate the yield of a book using ECC and redundancy by first calculating the yield of a column using Equation (2.22).

$$Y_{cecc} = (Y_{scecc})^{SizeOfRedundantColumn} \tag{2.22}$$

In Equation (2.22), Y_{cecc} stands for the yield of a column that uses ECC, and *SizeOfRedundantColumn* stands for the number of bits in a redundant column. Given the yield of the column, the yield of a book can be calculated using Equation (2.23).

$$Y_{ECCcrbook} = \sum_{j=0}^{n_{rc}} \binom{p+n_{rc}}{j} \cdot Y_{cecc}^{\,p+n_{rc}-j} \cdot (1-Y_{cecc})^{j} \tag{2.23}$$

In Equation (2.23), $Y_{ECCcrbook}$ stands for the yield of a book that uses ECC and column redundancy, p is the number of bits in the page and n_{rc} is the number of redundant columns. The yield of the DRAM can then be calculated using Equation (2.11) using Equation (2.23) as the yield of a book.

2.8.7. Yield Using Redundant Columns, Rows, and ECC

Calculating the yield of a DRAM chip that uses column redundancy, row redundancy, and ECC can be done by using most of the same equations that were used to calculate the yield with both row and column redundancy. These equations are Equation (2.16) through Equation (2.18). Equation (2.15) needs to be replaced with Equation (2.24) given below.

$$Y_{scecccr}^{\,bitsPerBook} = Y_{ECCcrbook} \tag{2.24}$$

In Equation (2.24), $Y_{scecccr}$ is an estimate for the yield of a single cell when ECC and column redundancy are used, and $Y_{ECCcrbook}$ stands for the yield of a book if ECC and column redundancy are used and is calculated in Equation (2.23). $Y_{scecccr}$ replaces Y_{sccr} in Equation (2.16) through Equation (2.18).

2.9 Applying Yield Equations to the IBM 16Mb DRAM

The equations and the analysis below was done for the 16Mb DRAM chip built by IBM, as described in [1] and [2]. There are two redundant columns per book with a page size of 128 bits and a column size of 2048 bits. There are four sections each of 4Mb with 24 redundant rows per section. The ECCword size is 137 bits total. There is one ECC word per book page. The values of all variables for the 16Mb IBM DRAM are given below in Table 2-4.

TABLE 2-4. Variable Values Using the IBM 16Mb DRAM

Variable	Redundancy Only	ECC
BitsInDRAM	$16 \cdot 1024 \cdot 1024$	$16 \cdot 1024 \cdot 1024 \cdot \frac{137}{128}$
SizeOfRedundantColumn	2048	2048

TABLE 2-4. Variable Values Using the IBM 16Mb DRAM

Variable	Redundancy Only	ECC
p	128	137
n_{rc}	2	2
NumberOfBooks	64	64
bitsPerBook	256K	274K
SizeOfRedundantRow	1024	1096
q	4096	4096
n_{rr}	24	24
NumberOfSections	4	4
ECCBits	-	137
ECCwordsPerBook	-	2048

We can then use the equations given in the previous sections to calculate the yield of the IBM DRAM chip with a changing average number of defects. This is shown below in Figure 2.11.

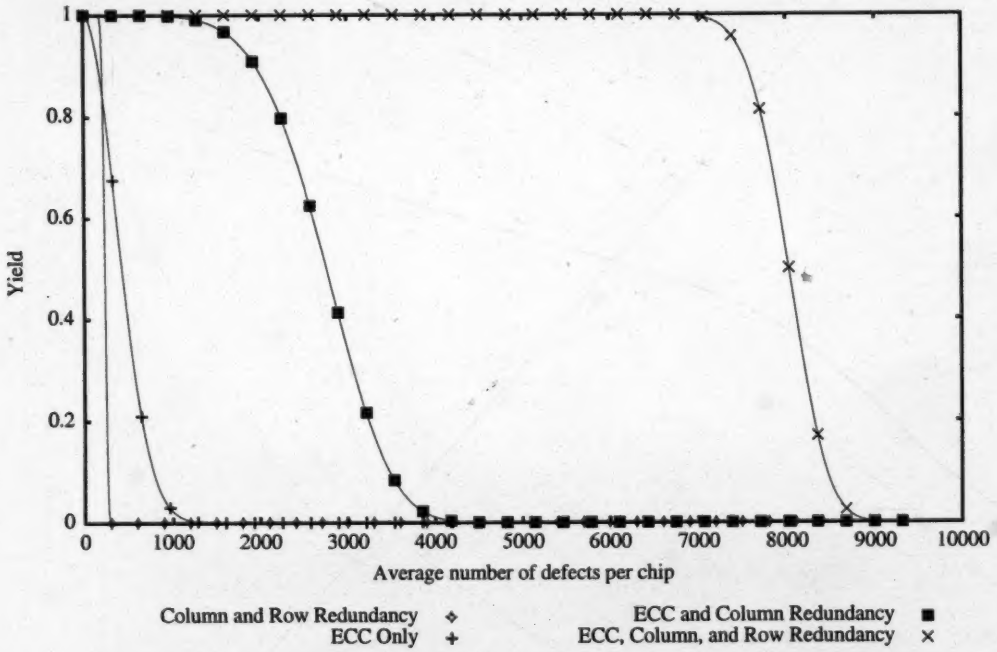


FIGURE 2.11. Yield of IBM 16Mb DRAM Using Redundancy and ECC

These results are similar to the published IBM results with the exception of the two new plots: column and row redundancy, and column and row redundancy with ECC. As mentioned earlier, conventional yield is not a good comparison of redundancy techniques

because it does not account for the area required by the redundancy. If we then replot this work using equivalent yield we can see the results below in Figure 2.12.

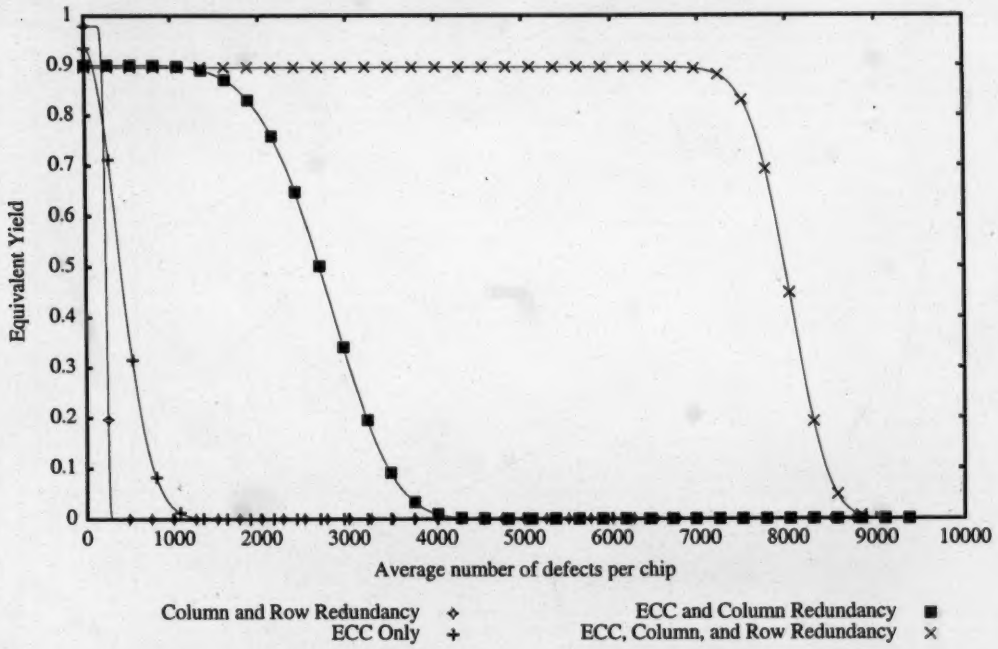


FIGURE 2.12. Equivalent Yield of IBM 16Mb DRAM Using Redundancy and ECC

The equivalent yield for column redundancy only and row redundancy only were not plotted in Figure 2.12 because the graph line was to close to the y axis to be

distinguishable with the given x axis scale. They are plotted with row and column redundancy in Figure 2.13.

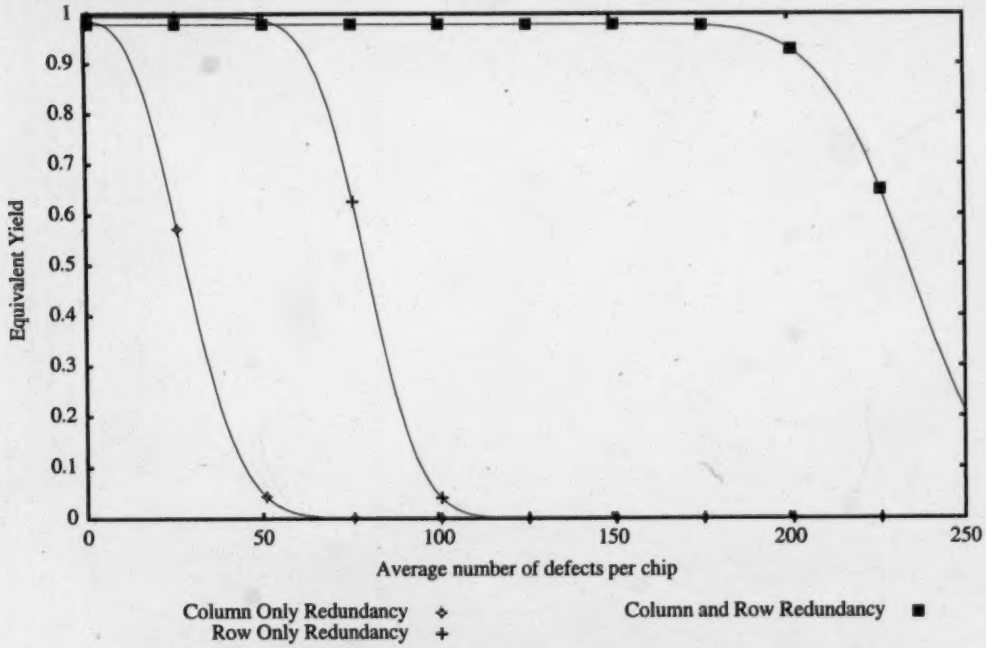


FIGURE 2.13. Equivalent Yield of IBM 16Mb DRAM for Column, Row, Column and Row Redundancy

The average number of tolerable defects for the IBM 16Mb DRAM chip at a 50% yield are listed below in Table 2-5. It was stated earlier that in [2] it was stated that the increase

TABLE 2-5. Average Number of Defects at a 50% Yield / Equivalent Yield

Redundancy	50% Yield (Figure 2.11)	50% Equivalent Yield (Figure 2.12)	Percent Area Overhead
Column Only	28	28	1.54%
Row Only	79	79	0.58
Row and Column	234	233	2.10%
ECC only	428	407	6.57%
ECC and Column	2761	2677	9.91%
ECC, Row, and Column	8027	7977	10.41%

in area due to the ECC was 11%. We assumed that there was a 2% increase in area due to the ECC circuitry requiring bit steering of the column bits. We can see from Table 2-5 that this 2% assumption seems appropriate because the ECC, row and column required 10.41%.

In a DRAM die the DRAM cells require decoders, sense amplifiers, and additional peripheral circuitry. The area that the DRAM cells take within the die is usually about 60% and is 63% for the 1Gb DRAM [27], and 49% for the 16Mb IBM DRAM[2] which means that there is a significant amount of non-memory-cell core circuitry. The amount of non-memory-cell core circuitry scales almost proportionally with the number of data bits, and therefore the ratio of additional area based on just memory is a valid comparison. It was assumed (and will be proven later in Chapter 5) that the area requirement for the ECC circuitry (the circuitry required to detect and correct the error) is negligible.

2.10 1Gb DRAM Example

These models can be extended to a 1Gb DRAM. This DRAM was chosen because it was the first demonstrated fully working 1Gb SDRAM and is described in [27]. The architecture and results are given below.

2.10.1. Architecture of a 1Gb DRAM

The 1Gb SDRAM is partitioned into 4 banks of 256Mb with 32 bits of data as output. Each bank is broken up into two sections of 128Mb and each section is broken up into four 32Mb sections. The redundancy is organized as 16 redundant columns per 32Mb section and as 64 redundant rows per 128Mb bank. This means that a *section* is defined as one of the 32Mb sections. The sections are built using 256Kb arrays with 256 bits on a bitline and 512 bits on a subword line. It was assumed that the 256Kb arrays are built from two subarrays aligned on the same wordline. From the published die photo it was determined that the sections are roughly square. Since the DRAM cells have an aspect ratio of 2:1, it was assumed that the 32Mb *section* architecture is 16x32 arrays. This means that since there are 16 redundant columns, and you need at least a two redundant columns per book (in order to fix shorted bitlines), so we will assume that the book will be two arrays wide by 32 arrays high with eight books per section. This gives a book a width of 2Kb and a height of 8Kb with two redundancy columns per book. This is a pessimistic assumption because it assumes a limited amount of flexibility of the columns. If the 16 columns could replace any column in the section then the yields would be better. Table 2-6 below gives all variable values for the 1Gb DRAM chip with an assumed size ECC word of 512 bits.

TABLE 2-6. Variable Values Using a 1Gb DRAM

Variable	Redundancy Only	ECC
BitsInDRAM	1Gb	$1Gb \cdot \left(\frac{523}{512}\right)$
SizeOfRedundantColumn	8Kb	8Kb
p	2048	$(512+11)\cdot 4$
n_{rc}	2	2
NumberOfBooks	64	64

TABLE 2-6. Variable Values Using a 1Gb DRAM

Variable	Redundancy Only	ECC
bitsPerBook	16Mb	$16Mb \cdot \left(\frac{523}{512}\right)$
SizeOfRedundantRow	16Kb	16Kb
q	8Kb	8Kb
n_{rr}	24	24
NumberOfSections	8	8
ECCBits	-	523
ECCwordsPerBook	-	32768

2.10.2. Calculating the Yield of a 1Gb DRAM with Redundancy

Using the same equations and the specific architecture of the 1Gb DRAM chip we can calculate what the yield will be when redundancy is used. The results of this are shown below in Figure 2.14.

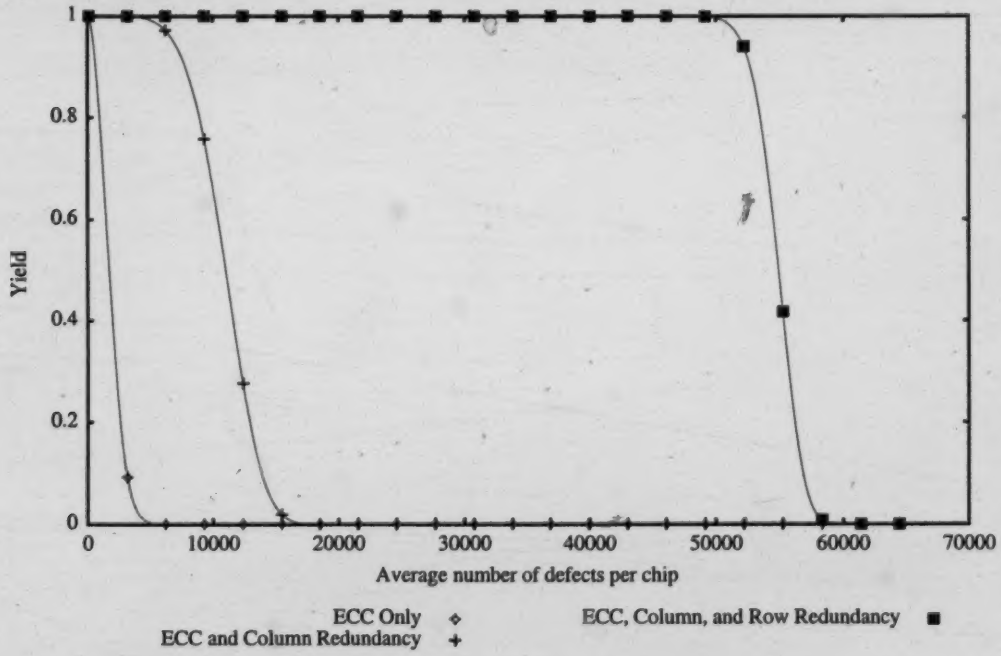


FIGURE 2.14. Yield of 1Gb DRAM using Redundancy and ECC

We can then convert the yield to an equivalent yield and use this as the comparison of redundancy methods. This result is shown below in Figure 2.15.

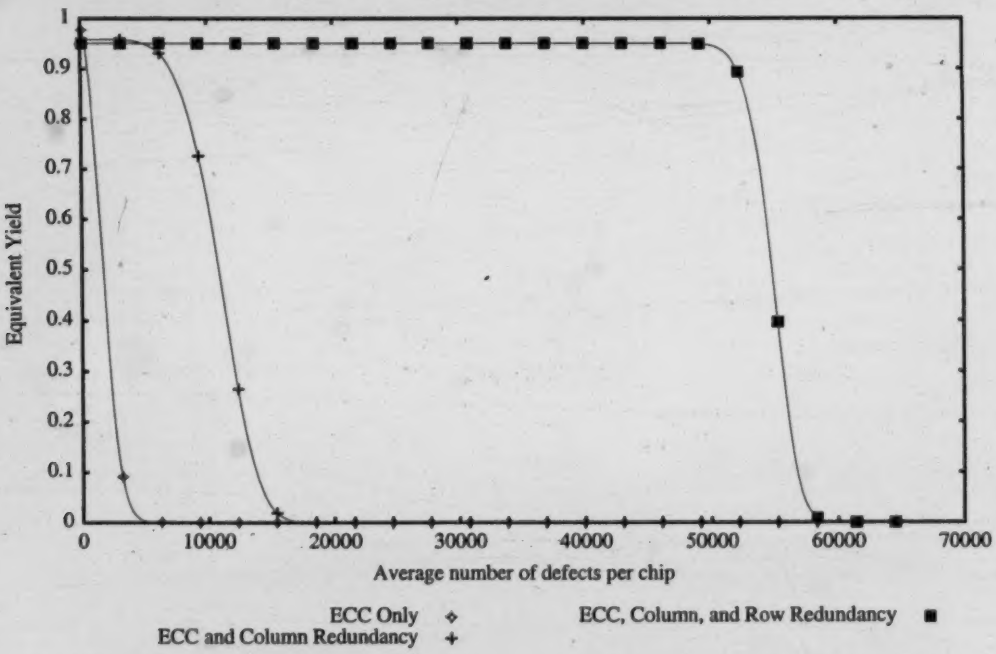


FIGURE 2.15. Equivalent Yield of 1Gb DRAM Chip Using Redundancy and ECC

We can see from this figure that, when the average number of defects is low, it is better to have ECC and column redundancy instead of ECC, column, and row redundancy because the equivalent yield is higher. The advantage that all three redundancies give is that an equivalent yield is still achieved when the average number of defects is high. Column only redundancy and row column redundancy were not plotted on this figure due

to their 50% yield points being so small. Instead they are plotted by themselves below in Figure 2.16.

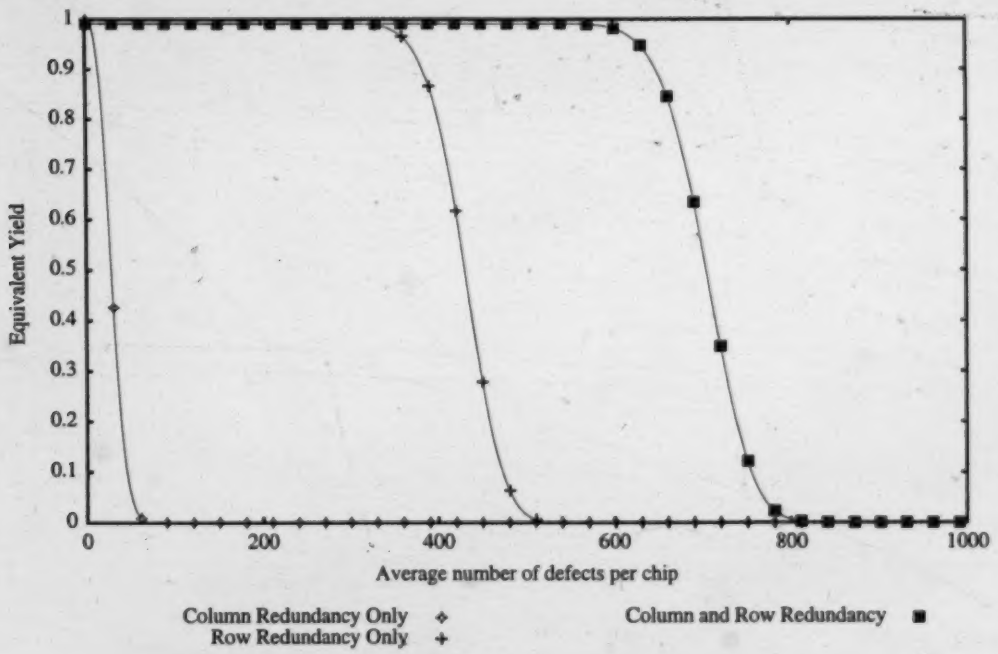


FIGURE 2.16. Equivalent Yield of 1Gb DRAM for Column, Row, Column and Row Redundancy

The average number of tolerable defects at a 50% yield point for both yield and equivalent yield are given in Table 2-7 below.

TABLE 2-7. Average Number of Defects at a 50% Yield / Equivalent Yield

Redundancy	Yield (Figure 2.14)	Equivalent Yield (Figure 2.15)	Percent Area Overhead
Column Only	28.5	28.5	0.098%
Row Only	430	430	0.78%
Row and Column	705	705	0.82%
ECC only	1707	1680	2.1%
ECC and Column	10997	10866	4.20%
ECC, Row, and Column	54931	54751	4.94%

2.10.3. Equivalent Yield of a 1Gb DRAM with Different ECC Word Sizes

In the above figures the ECC word size was chosen to be 1/4 the width of the book, i.e. 512 bits. In Figure 2.17, below, the different ECC word size was changed and the average number of tolerable defects at a 50% yield point was plotted. The word size was varied from 16 data bits to 2048 data bits.

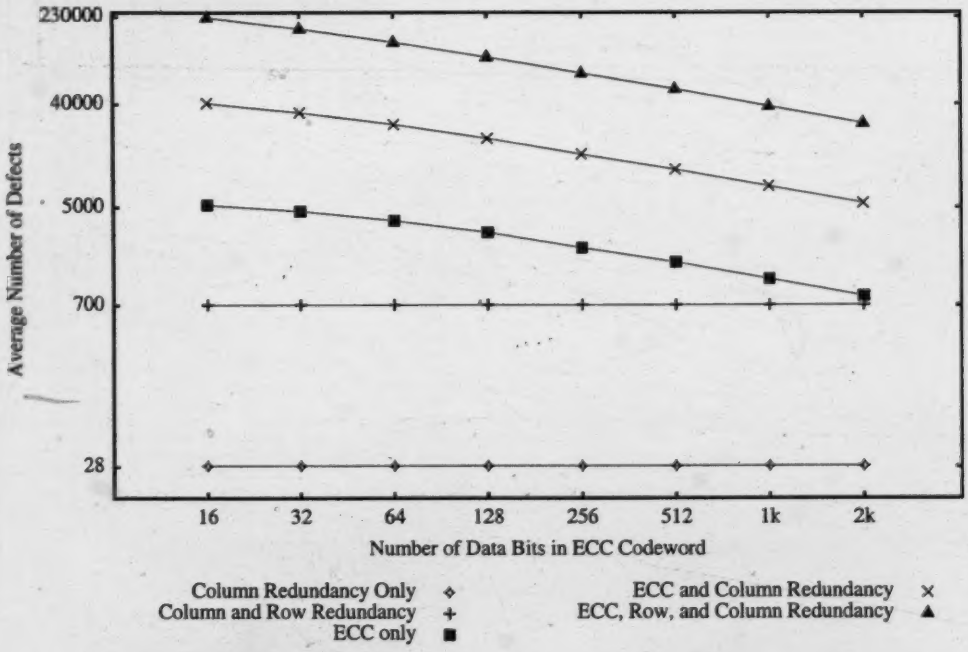


FIGURE 2.17. Average Number of Tolerable Defects at 50% Equivalent Yield

There are five lines in the figure. They are the yield using only column redundancy, column and row redundancy, using only ECC, using ECC and column redundancy, and using column, row, and ECC. The equivalent yield for only column, and column and row redundancy are horizontal because they do not use ECC. It is shown in Figure 2.17 that synergistic fault tolerance effect still exists for all of the ECC word sizes. It is also shown that as the ECC word size becomes larger, the number of tolerable defects goes down, as expected. This occurs because, as the word becomes larger, less errors can be tolerated. The figure does not examine any ECC word sizes larger than 2048 bits. This is because the width of the book is 2048 bits. There is no reason that the ECC word size could not be larger than a book but there is a limitation that all of the bits must be available to the ECC circuitry. The limitation to the maximum size of the ECC word will be the effectiveness of the ECC and the maximum DRAM page size.

2.11 Summary

Hamming codes can be used to correct hard errors and thus increase the yield. Hamming codes were described and analyzed for size and speed performance. Equivalent yield was used as a way of comparing redundancy techniques. Equivalent yield accounts for the area required by the redundancy which the yield does not. Very powerful fault synergism can be obtained by combining row redundancy, column redundancy, and ECC. Column redundancy, row redundancy, and ECC were extended to a more modern 1Gb DRAM chip. Perhaps other methods for redundancy (such as block marking) will be more effective at raising the equivalent yield in the presence of relatively large number of defects.

Chapter 3: Bad Block Marking Applied to Memory

It was mentioned earlier that hard disk drive manufacturers mark the bad sectors of the recording media, then sell the drive at its nominal capacity. This chapter extends this notion of bad block marking to solid state memory in order to increase the equivalent yield (or lower the cost per working bit). We investigate how bad block marking (BBM) information can be stored, design techniques, and possible implementations. Two of the schemes examined are to use a bitmap to mark memory and to use a list representation. The 1Gb DRAM chip example from Chapter 2 will be used for most of the bad block marking analysis. Using both BBM and traditional redundancy techniques is also investigated in order to determine when and how the use of BBM may be beneficial.

3.1 Non-Volatile Memory for Storing a List or Bitmap

Bad block marking requires some type of non-volatile memory in order to store the bad block information. The manufacturing test data is required for reasons described in Section 3.3. Therefore we need some kind of non-volatile memory that could be built in a DRAM chip using a DRAM process. A single poly EPROM is described in [24] and could be built using a DRAM process. We approximate the non-volatile memory cell size as 10 times the size of a DRAM cell. In [27] a 1Gb DRAM was built with a cell area of $0.334\mu m^2$ using a $0.16\mu m$ process. The EPROM cell described in [24] was built using a $3\mu m$ process with a cell area of $1188\mu m^2$. Assuming that all aspects of the design scale linearly, we get a cell size of $3.38\mu m^2$ in the $0.16\mu m$ process, which is approximately 10 times larger than a DRAM cell. We are assuming that the yield of the non-volatile memory will be 100%.

Current DRAM manufacturing uses an array of fuses to control the redundant row and columns within the memory. It should be noted that fuses are a type of non-volatile memory and could also be used for bad block marking. But since the size of a fuse cannot shrink with the process feature size (because a laser needs to be positioned mechanically to focus on the fuse cutting region) we propose that the fuses be replaced with a device that does shrink with feature size, namely, EPROM, EEPROM, or flash memory. It should also be possible to use the same memory for both the row, column redundancy, and for the BBM list, thus increasing the equivalent yield even further.

3.1.1. A Negligible Die Overhead Method for Marking Bad Blocks

A very simple technique that involves negligible cost on the DRAM die would be to include a readable serial number on each die sold. This serial number could then be used to query a database that contains a list of the bad bits. The list could then be incorporated into a system using the file memory by either storing it on a hard drive or in flash memory.

3.2 Bad Block Marking

As mentioned already, bad block marking applies techniques already used for imperfect media storage devices, such as hard drives, to solid state memories. They are applied using a type of non-volatile memory on the memory and by using various marking techniques. These techniques are examined in detail and compared to traditional techniques.

3.2.1. Partially Good Products Method

The equivalent yield as defined in Section 2.3 is the yield of the chip multiplied by the fraction of the data bits over total number of bits. The yield of the chip is calculated as the yield with the use of redundancy or the yield considering only partial functionality of the chip. In Section 2.3, it was the yield of the die after the use of redundancy multiplied by the fraction of usable bits. The partially good products technique [22] is a technique that can be used to calculate an equivalent yield if partially functional chips are used. In the partially good products technique, the equivalent yield is the yield of the partially good product multiplied by the fraction of the product that is good. Calculating equivalent yield using the partially good products technique and calculating the equivalent yield, as presented in Chapter 2, are actually the same because they both represent functional bits over the total area.

Basically, when using the partially good products technique, the yield of the partially functional chip is multiplied by the fraction of the chip that is functional. This is done so that when the yield is high and all of the die is functional then we have a very high equivalent yield. The method used for partially good products is taken from [22] and calculates the equivalent yield using Equation (3.1).

$$Y_{eq} = Y_{ag} + \sum_{n=1}^{k-1} \frac{n}{k} Y_{pg_{\frac{n}{k}}} \tag{3.1}$$

In Equation (3.1) Y_{ag} refers to the yield of an all good chip, $Y_{pg_{\frac{n}{k}}}$ refers to the yield of a partially good chip, n refers to the number of sections that are functional in the partially good chip, and k refers to the total number of independent sections in the chip. When the fraction of the die that are functional ($\frac{n}{k}$) goes to zero, so does the equivalent yield (Y_{eq}) (even if the yield ($Y_{pg_{\frac{n}{k}}}$)is still high). This is because it does not matter if the yield is high if there are no functional parts of the chips. The all-good yield can be calculated as shown in Equation (2.8) in Chapter 2. Equation (3.1) calculates the equivalent yield as the sum of the yield of all partially good products times the fraction of the products that is good. The

partially good yield can then be determined for any n good sections out of k total sections as shown below in Equation (3.2).

$$Y_{pg_{\frac{n}{k}}} = \binom{k}{n} Y_p{}^n (1 - Y_p)^{k-n} \tag{3.2}$$

In Equation (3.2) Y_p is the yield of a page or block. Note that the sum of all of the possible number of errors can be added up giving the equivalent yield (Equation (3.3)) below. This equivalent yield is expressed as the sum of the all-good yield and all partially good yields. This equation simplifies to the yield of a single page because we are summing up all of the combinations for the partially good product. When the yield of all partially good products and the yield of the all good product is summed it will be the yield of a page because we are summing from one page being good to all pages being good.

$$Y_{eq} = Y_{ag} + \sum_{n=1}^{k-1} \left(\frac{n}{k}\right)\binom{k}{n} Y_p{}^n (1 - Y_p)^{k-n} = Y_p \tag{3.3}$$

The sum goes from 1 good block to k-1 good blocks. All blocks being good (k) is accounted for in Y_{ag}. Simplifying the equation by having the sum go from 1 to k was not done because the equation has already been simplified to the yield of a page (Y_p).

The equivalent yield of the chip can then be calculated using the partial product equivalent yield as shown in Equation (3.4).

$$Y_{chipeq} = \left(\frac{ds \cdot Y_p}{ds + nv}\right) \tag{3.4}$$

In Equation (3.4) ds is the number of data bits in the DRAM, and nv is the amount of non-volatile memory. Both of these are measured in bits.

3.2.2. Motivation for Bad Block Marking

If we limit ourselves to block accesses, then we can remove the low latency constraint and the need to have a continuous memory space. The added flexibility should offer opportunities for improving on the conventional row and column redundancy found in DRAM chips. Assuming that not all blocks need to be functional in order to have a functioning die means that the yield can be increased. File memories could map around bad blocks (slightly reducing total capacity) just as hard drives keep track of faulty sectors. It was mentioned earlier that there are at least two methods of marking a faulty block as defective: a faulty block map or a faulty block list. A block map uses a bitmap to represent blocks and a list uses the address of the block. A file memory approach may offer an attractive way of effectively salvaging many more partially functional chips, thereby significantly improving the economics and accelerating the introduction of gigabit technology.

There are also good reasons for using bad block marking and keeping the traditional row and column redundancy. The manufacturer still has the goal of manufacturing DRAM so we would still want to continue to use the row, and column redundancy so that 100% continuous DRAM could be obtained. This idea will be analyzed further in Section 3.6.

3.3 Bad Block Marking Techniques

It was already stated that some type of non-volatile memory is required for bad block marking. The most obvious solution for implementing bad block marking would be to make no changes to the DRAM design and to sell the chips as they are. The DRAM chip would then be sold as file memory if it had less than 100% capacity. Then the system that is using the file memory could do a memory test and determine which cells are bad and use an appropriate data structure to keep track of the bad regions. This scheme fails because the testing done during manufacturing uses higher temperatures and tests over the entire voltage range. Tests that verify retention time are also done. These tests cannot be easily performed once the memory is in the system. This means that either bad cells need to be identified when the chip is manufactured hence the need for some type of non-volatile memory, or a separate file needs to be associated with each die.

The techniques used to mark bad blocks will depend of the types of faults that are encountered during manufacturing. The techniques covered in Section 3.4 and Section 3.5 describe the techniques required to mark bad cells (including wordlines and groups of cells) and bitlines. Section 3.4 specifically describes how to mark bits or groups of bits as bad. This can range from 1 bit, sub-section, section, bank, or wordline being marked as bad. Section 3.5 describes a technique that can be used to mark bad bitlines. With the use of these two techniques, any combination of can also be marked, such as a wordline to bitline short. For the purpose of this analysis, an assumption was made that every block is the same size and only one technique is used at any one time. It was also assumed that the hard defects do not change over the lifetime of the DRAM chip. A few other possible methods and combinations for marking are described in Section 3.4.3.

3.4 Marking Cells as Bad

3.4.1. Bitmap Marking

As mentioned earlier, one method that can be used for marking blocks as bad is to use a block bitmap. Each bit in the bitmap represents one page (or one block). If any bits in a page are bad then the corresponding bitmap bit is set to indicate that the block is bad. The effectiveness of this method depends on the block size and the number of bits that are still functional after marking (remember that we need working bits). As the block size decreases by some factor we can recover more functional bits, but the capacity of the bad block bitmap must increase by the same factor. The block size will be limited by the amount of non-volatile memory that would be on the memory. The amount of fixed non-

volatile storage determines the size of the blocks because the smallest block size will always be used. We can calculate the size of a bitmap block by dividing the DRAM size by the number of non-volatile bits.

3.4.2. List Marking

Another scheme that can be used for bad block marking is a block list. This data structure lists the starting address of each bad block. As the block size increases in size, the number of address bits required to mark a block decreases, the number of blocks that can be marked increases, and the number of bits that can be marked as bad increases. The block can vary in size from 1 bit to a maximum size governed by Equation (3.5).

$$\log_2\left(\frac{ds}{bs}\right) = lb \tag{3.5}$$

In Equation (3.5) *ds* is the number of data bits in the DRAM, *bs* is the number of bits in a block when list marking is used, and *lb* is the number of bits required to mark the list block. The maximum number of blocks that can be marked is limited by the amount of non-volatile memory and is given in Equation (3.6)

$$\frac{nv}{lb} = MaxBlocks \tag{3.6}$$

In Equation (3.6) *nv* is the number of non-volatile bits. The total number of blocks in the DRAM is the size of the DRAM divided by the size of a block. This is shown in Equation (3.7).

$$\frac{ds}{bs} = NumberOfBlocks \tag{3.7}$$

Since we would never want to mark more blocks than exist we can set Equation (3.6) equal to Equation (3.7) and rearranging we get Equation (3.8)

$$\frac{nv \cdot bs}{ds} = lb \tag{3.8}$$

Now we can equate Equation (3.5) to Equation (3.8) and get Equation (3.9).

$$\log_2\left(\frac{ds}{bs}\right) = \frac{nv \cdot bs}{ds} \tag{3.9}$$

Since *ds* and *nv* are usually fixed we can solve for *bs*. The maximum block size using a 1Gb DRAM with 128Kb of non-volatile memory is a block size of 108707 bits or 64Kb if a power of 2 is being used. The block size for list marking can be larger than the size of a bitmap block. It only becomes beneficial to use a list block size equal to or larger than the bitmap block size if the alignment used for the list is not the same as the alignment for the bitmap. List block sizes larger than the bitmap block size will not be investigated further.

It is interesting to note here that at very high number of defects it is actually more efficient to go back to list marking. This will be explained later. The effectiveness of this scheme is now limited by the size of the bad blocks and by the number of non-volatile bits for storing the list. An analysis of the optimal list size is given below.

Calculating the Equivalent Yield of a Block

Recall that the equivalent yield for the chip using partially good products is the yield of a block (Equation (3.4)). We can calculate the yield of a block by using Equation (3.10).

$$Y_{block} = Y_{sc}^{bs} \tag{3.10}$$

where Y_{block} is the yield of a block, Y_{sc} is the yield of a single cell (as calculated from Equation (2.7)), and bs is the block size. This is the equivalent yield when bad block marking is used.

When a bitmap is used the size of a block is fixed and marking can always be accomplished. When a list is used the size of the block can vary and marking can only be accomplished if there is enough non-volatile memory to store the addresses of the bad blocks. These sizes and choices are examined next.

Optimal Block Size for Bad Block Marking List

The 1Gb DRAM chip will be used for all of the optimal bad block marking size analysis. Determining the optimal block size for a bad block marking list depends on a few independent factors. These factors are the average number of defects present in the DRAM and the number of non-volatile bits that can be used for storing the list. It was assumed that there would be a fixed amount of non-volatile memory in the DRAM. So the size of the non-volatile memory was fixed at 128Kb and the average number of defects was varied and the equivalent yield was plotted. This size of non-volatile memory was chosen by trial and error and this size can tolerate approximately 90000 errors with an area overhead of only 0.12%. The maximum block size using a bitmap is 8Kb and is calculated as 1Gb/ 128Kb. The results are shown below in Figure 3.1. Since the number of defects would be

known for each die at the point of manufacturing, then one could calculate the optimal block size for each DRAM die and then use that best block size.

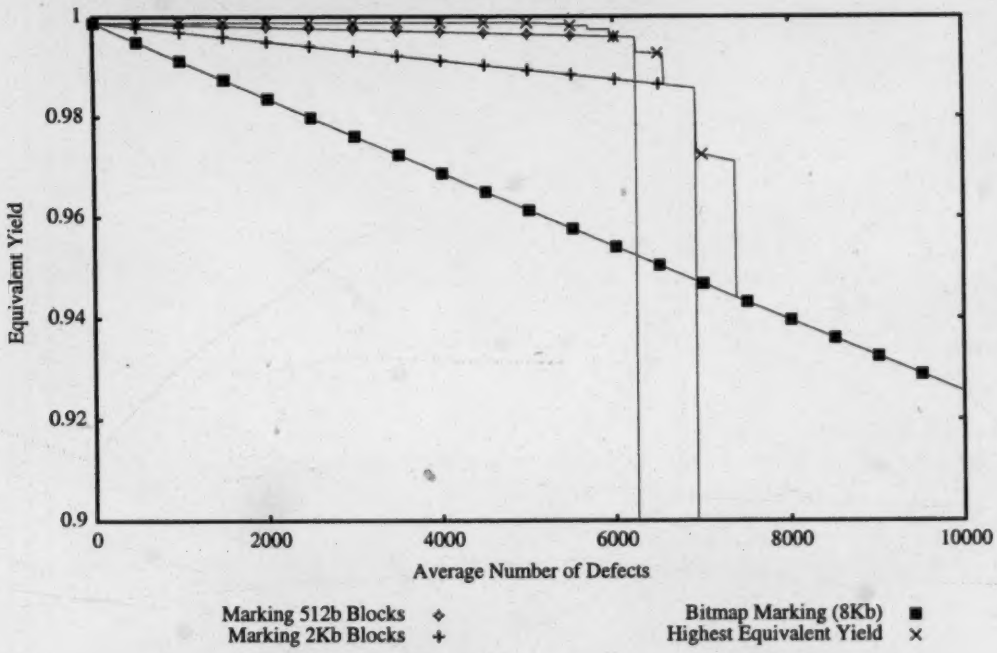


FIGURE 3.1. 128Kb of Non-Volatile Memory for Bad Block Marking List Using 1Gb DRAM

We can see from the figure that, if there is a low number of defects, the block size should be small. As the average number of defects increases, the optimal block size also increases. If the block size for list marking reaches 8Kb then list marking performs identically (for low average defect numbers) as a bitmap does. The difference is that for a high average number of defects, the list marking method will fail because it requires more bits to mark the blocks than are available. We can also see in the figure that the highest equivalent yield is indicated at every average number of defects. This gives us the optimal solution for every possible average number of defects.

We can then look at the effect that a much smaller number of non-volatile bits has on the equivalent yield by replotting the graph using 256 bits of non-volatile memory. There are a few reasons why we may want to examine the use of a very small number of non-volatile bits. The first is that the area of 256 non-volatile bits compared to the area of 1Gb is 0.002% (very modest). Therefore, the cost for including them should also be almost zero. The second major reason is that if there are relatively few errors, then they can be marked using a list successfully within the 128 bits. If there is a large number of errors, chances are that they are not random single failing errors but rather cluster defects and can most likely be marked using redundancy in combination with bad block marking.

Figure 3.2 below shows the equivalent yield for a different average number of defects if 256 bits are used for the block marking.

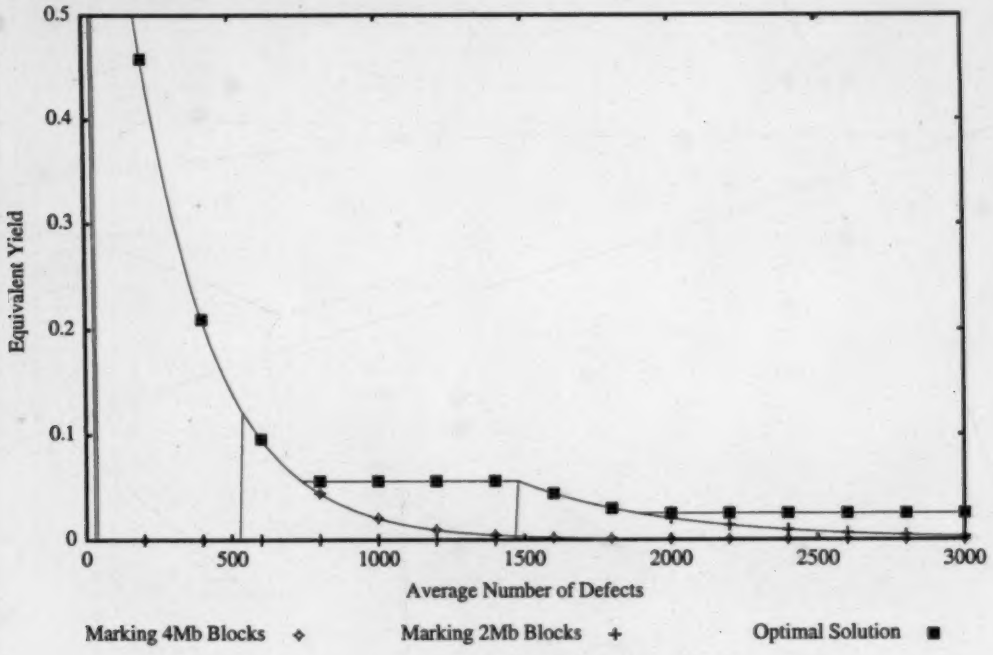


FIGURE 3.2. 256 Bits of Non-Volatile Memory for Bad Block Marking

We can see from the figure that at a certain average number of defects the equivalent yield increases from zero. This seems very counter-intuitive at first because, it does not seem to make sense that we would get a higher equivalent yield with more defects. As the average number of defects goes from low to high, the list marking goes through three stages. The first stage is that the bits that are required to mark the bad blocks is less than the amount of non-volatile bits and list marking can be used. The second stage is that the number of bits required to perform list marking is larger than the non-volatile memory. Therefore, list marking fails and bitmap marking must be used. The last stage is when the number of bits that are required to mark the good blocks as good is less than the amount of non-volatile memory, so in this way we can get an equivalent yield. This means that if we have a small amount of non-volatile memory, and a large number of average defects (for the amount of non-volatile memory), it may be more beneficial to consider good bits as bad, decrease the block size and use good block marking.

Results of List Marking

From the above results we can see that using a bitmap to mark bad blocks is only beneficial if it is not possible to use a list marking method (bad block marking, or good block marking). For a high average number of defects it is more beneficial to consider good bits instead of bad and to use good block marking. This corresponds to taking the peaks on the graph and extending them left until they intersect the previous larger block size equivalent yield. At the points where either a bitmap or a list could be used it is more beneficial to use list marking. If a bitmap is used the address of each block is assumed to be aligned with the block sizes, where, since the list stores the address of the start of the block, the list is more flexible in it's marking.

Effect of Number of Bits for BBM on the Equivalent Yield

We can examine how the amount of non-volatile memory effects the equivalent yield at different defect rates. These results are shown below in Figure 3.3. Each of the lines in the plot represents the optimal block size for any average number of defects using list or bitmap marking. The equation used for this figure is Equation (3.10) where the optimal marking scheme is chosen (bitmap or list) and if a list scheme is chosen then the optimal

power of two block size is used. This is how all of the bad block marking equivalent yields are calculated in all plots that use bad block marking.

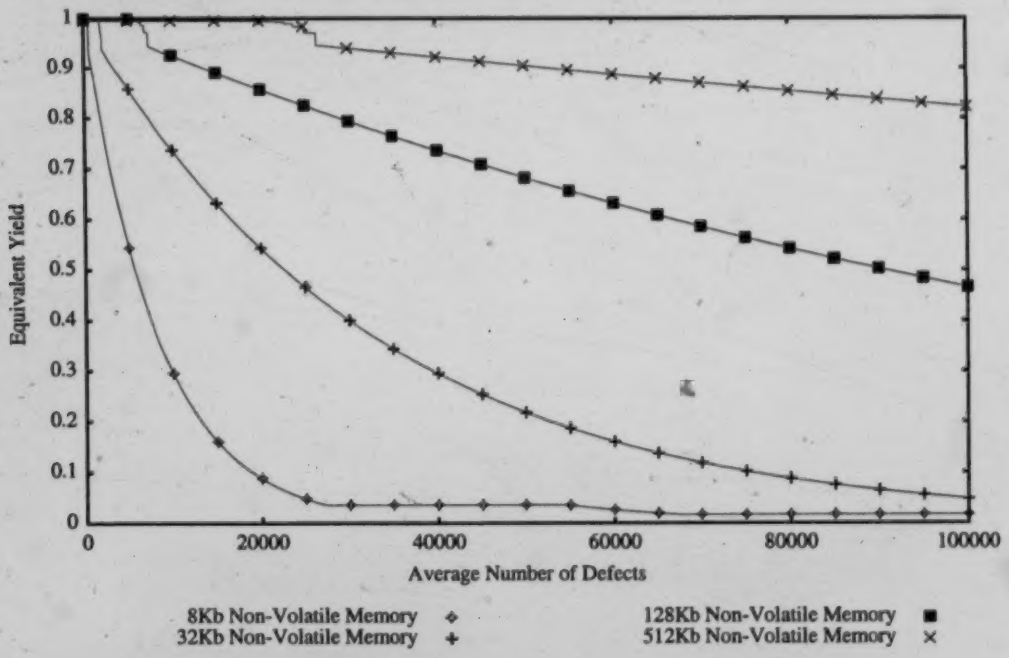


FIGURE 3.3. Bad Block Marking List Technique with Different Non-volatile Memory Sizes

The equivalent yield remains high until the number of BBM bits is no longer sufficient to hold the entire list. At this point the bitmap marking technique is used. After the equivalent yield becomes quite low it then becomes more beneficial to go back to block marking and use good block marking. The 50% points and the area overhead required for the non-volatile memory are given below in Table 3-1.

TABLE 3-1. 50% Equivalent Yield Points for Different Amounts of Non-Volatile Memory

Amount of Non-Volatile Memory (bits)	Average Number Of Defects	Percent Area Overhead
256	177	0.0002%
512	354	0.00048%
2K	1419	0.0019%
8K	5677	0.0076%
32K	22703	0.031%
128K	90692	0.12%
512K	360855	0.49%
1M	7.20×10^5	0.967%

TABLE 3-1. 50% Equivalent Yield Points for Different Amounts of Non-Volatile Memory

Amount of Non-Volatile Memory (bits)	Average Number Of Defects	Percent Area Overhead
2M	$1.4x10^6$	1.92%
4M	$2.75x10^6$	3.76%
8M	$5.18x10^6$	7.25%
16M	$9.20x10^6$	13.51%

Comparing the 50% equivalent yield points of 2K BBM bits to row, and column redundancy (from Table 2-7) we can see that row and column redundancy can tolerate an average of 705 defects with an area overhead of 0.82% whereas bad block marking can tolerate an average of 1419 defects with an overhead of 0.0019%. Bad block marking is approximately 432 times smaller and can tolerate twice as many defects. It should be noted that the non-volatile memory required to indicate the block size and the BBM technique is not included in the amount of non-volatile memory. In the worse case (256 bits of non-volatile memory) the bits required to indicate the size and technique is 5. With 5 bits there are a total of 32 possible combinations, one is use to indicate bitmap marking, and the other 31 are used for indicating different list sizes. This means that the list size can vary from one bit to the size of the DRAM if powers of 2 are used. Five bits is approximately 0.02% of 256.

3.4.3. Combination Marking

List marking and bitmap marking can be combined to allow even greater flexibility. Both the address of the initial bit that is bad and the size of the block could be stored. This would allow different block sizes to be accommodated at the same time. The size would most likely be coded into as few bits as possible with the codes being set by the expected patterns of manufacturing defects. For example, if there were three types of defects (failing cells, wordline failures, and wordline shorts) then the three possible sizes could be coded using two bits representing a single bit, a wordline and two wordlines. With this combination marking method, many different patterns of faults can be marked efficiently at the same time within the non-volatile memory. The way in which the combination marking would be done would depend on the specific architecture of the DRAM and would also depend on the faults that are present in manufacturing. A more detailed analysis of combined techniques was not done because it is architecture and fault-dependent and is best left as future research. It seems very likely, however, that the use of some type of combination marking scheme should result in a higher equivalent yield.

3.5 Marking Bitlines as Bad

Our benchmark 1Gb DRAM has 256 cells on a bitline with 30 bits required to mark any single bit of the 1Gb. To mark any bitline would require 22 bits. If a bitline fails then the

resulting bad bits could be recorded as a number of single cell failures, where the number of failures is equal to the number of cells on a bitline. However, this is a very inefficient method for dealing with a bitline failure because it would require 30 address bits for each cell times 256 failing cells or 7680 bits. A bitline could be marked bad by just specifying the beginning address of the bad bitline. We could then calculate the addresses of all other bits on the faulty bitline by adding offsets of zero to 255 to the row address and applying address scrambling if necessary. If there is a row scrambling function then it would be used when incrementing the row address. The bitline marking technique would require 22 bits of non-volatile memory to mark the bitline as bad, which is 349 times less area than the area required to mark all affected bits.

The software model used for bad block marking is described later in Chapter 4. It uses the beginning address of the bad block and specifies the number of bad bits following that address. This shows that specific block marking methods can be designed that allow more efficient use of the non-volatile memory. The most efficient method designed will depend on the fault model for the DRAM and its architecture.

3.6 Using Bad Block Marking with DRAM

We can now examine how bad block marking compares to traditional row and column redundancy techniques. The deployment information for redundant rows and columns in conventional DRAM is stored in arrays of fuses, which are selectively blown using a laser. Any memory that contains redundant or bypassed elements must also contain a mapping, stored in non-volatile memory (fuses are a form of non-volatile memory), that indicates where the defects are and how the elements are to be used. Such a "defect list" for hard disks can be kept on the disk itself since the medium is non-volatile. Bad block marking could and probably should use the same non-volatile memory as row and column redundancy. As mentioned earlier, EEPROM or flash memory should be used because it scales with the process.

3.7 Applying Bad Block Marking to the IBM 16Mb DRAM

The techniques described above for bad block marking can now be compared to traditional redundancy schemes in greater detail. The techniques described above will first be applied to the IBM DRAM to see how BBM affects the yield. Plotting the equivalent yield versus the average number of defects, we can see the results of using ECC only, redundancy only, redundancy and ECC, and bad block marking. Figure 3.4 shows the results for using bad block marking in the IBM 16Mbit DRAM. The equivalent yield for bad block marking for Figure 3.4, Figure 3.5, and Figure 3.6 are all the same being plotted

with different x axis ranges. They all use 128Kbits of non-volatile memory for the bad block marking.

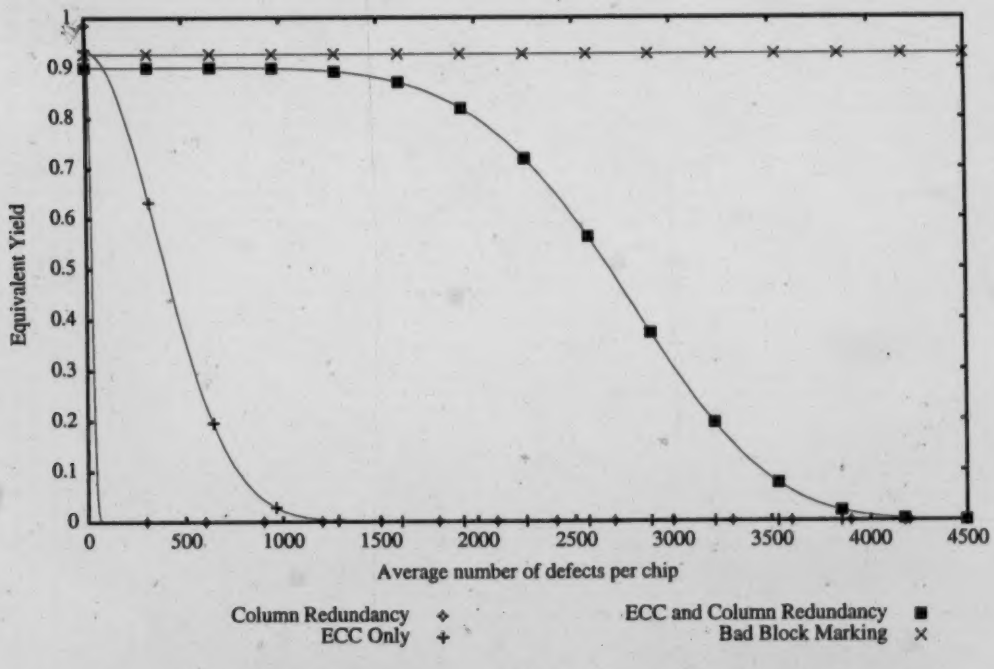


FIGURE 3.4. Equivalent Yield Using Bad Block Marking in a 16Mb DRAM

The equations used to calculate the equivalent yield of column, ECC, and ECC and column redundancy are given in Chapter 2 using most of the equations between Equation (2.7) and Equation (2.23). The equation to calculate the equivalent yield of bad block marking is Equation (3.4). We can see that this plot is very similar to Figure 2-1 except that bad block marking has been included and equivalent yield is being used instead of conventional yield. We can see from the plots that marking bad blocks reduces only very slowly the cost per working bit over an increasing average number of defects. The slope of the line depends on the granularity of the markable block size. If the granularity is very fine (i.e. we can mark single defects as bad without any cost for marking), then the equivalent yield of the die would be very close to the yield of a single bit. Remember from Equation (3.4) that the equivalent yield is a function of the page yield. If the page size becomes the size of a bit, then the equivalent yield becomes the yield of a single bit adjusted by the size of the DRAM and the non-volatile memory. However, this is impractical due to the area required by the non-volatile memory. In our example the largest block size was 128 bits (using 128Kb of non-volatile memory). Using this size gives us the results shown in Figure 3.4. Table 2-5 has the average number of defects of 28 defects for only column redundancy, 407 defects for only ECC, 2677 defects for column redundancy and ECC. The average number of defects for bad block marking is 81005 at the 50% equivalent yield point.

Figure 3.5 below shows how the equivalent yield is affected by bad block marking over a larger average number of defects. This figure also includes the equivalent yield of various combinations of ECC, Column, and row redundancy.

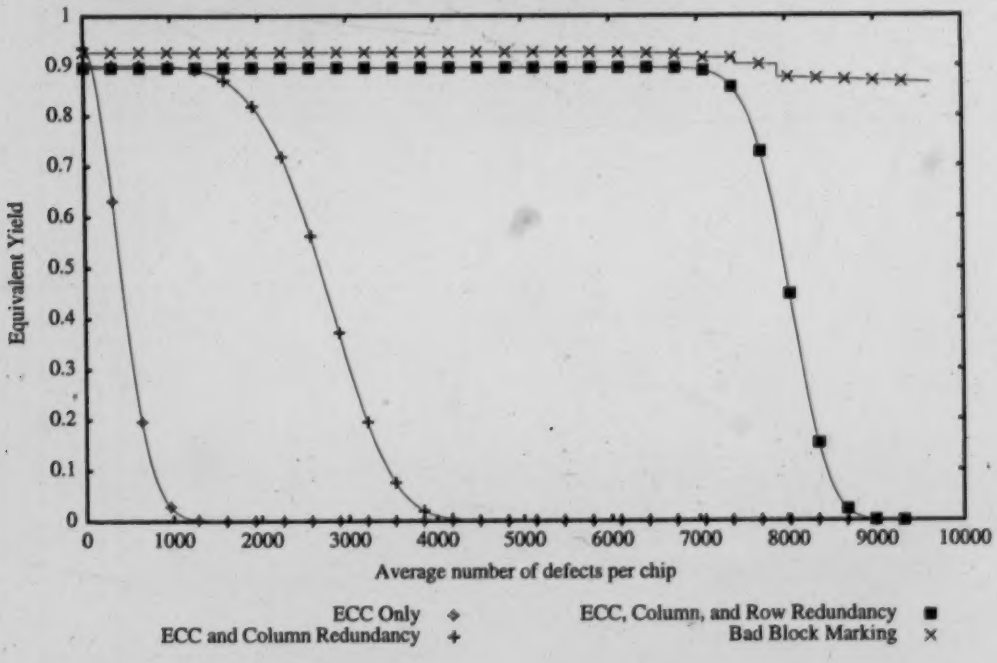


FIGURE 3.5. Equivalent Yield Using Bad Block Marking in a 16Mb DRAM (continued)

We now examine how bad block marking is affected by larger average numbers of defects. A plot of how the equivalent yield is affected when using bad block marking for increasing the average number of defects is shown below in Figure 3.6.

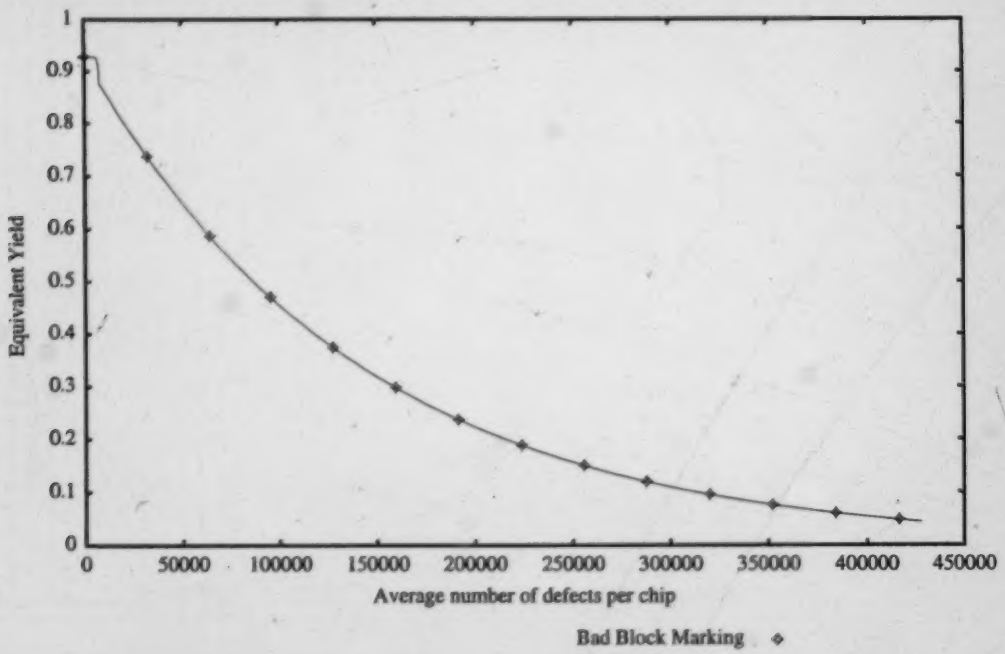


FIGURE 3.6. Equivalent Yield Using Bad Block Marking, High Defects

Recall that the knee in the curve if from switching from list marking to bitmap marking and can be seen on many different sizes of non-volatile memories in Figure 3.3. It was stated above that the 50% equivalent yield point is at 81005 defects which is much larger than any of the conventional redundancy techniques in DRAM.

3.8 Bad Block Marking Applied to the 1Gb DRAM

We have already seen how the equivalent yield for the 1Gb DRAM is affected when using bad block marking (Figure 3.1, Figure 3.2, and Figure 3.3). They show how the equivalent yield changes with different sizes of non-volatile memory while varying the average number of defects. Now we can see how BBM compares to the redundancy techniques examined in Chapter 2. We can also extend BBM and see how the equivalent yield is effected when used in combination with row and column redundancy.

3.8.1. Bad Block Marking

The redundancy techniques that were already examined for the 1Gb DRAM chip were row redundancy, column redundancy, and ECC. Assuming 128Kb of fixed non-volatile memory for the DRAM, the equivalent yield for BBM is shown below in Figure 3.7. The equations to plot all equivalent yields except bad block marking are taken from Chapter 2, and the equivalent yield for bad block marking is calculated using Equation (3.10).

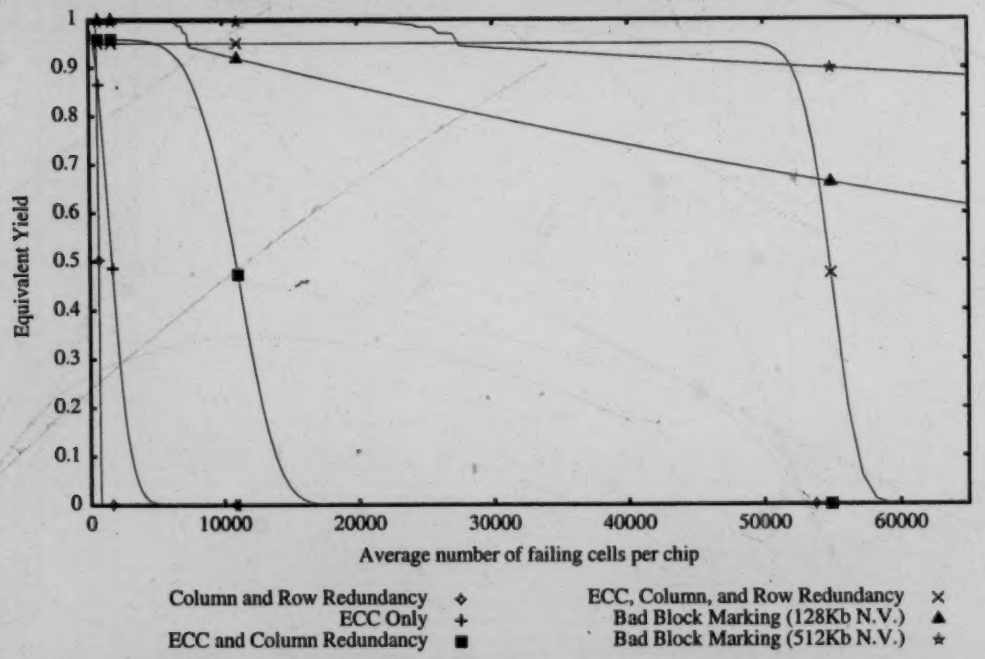


FIGURE 3.7. Equivalent Yield Using Marking Bad Blocks in a 1Gb DRAM

The fifty percent yield points for this figure are given in Table 3-1 and Table 2-7. The figure shows us that BBM gives higher equivalent yields than the alternatives for all average numbers of defects except when ECC, column, and row redundancy is used together. BBM does, however, perform better for lower number of defects and for larger number of defects. One very important thing to note about the figure is that the equivalent yield is very high for a low number of defects for BBM. The use of row, column, and ECC has an area overhead of approximately 4.94% where BBM has an overhead of approximately 0.12%.

3.8.2. Bad Block Marking with Row and Column Redundancy

As mentioned earlier, there are a few reasons why we would want to use bad block marking and traditional row, and column redundancy together in the same memory chip. If only file memories were being manufactured, then redundancy would not be used because perfect continuous media is not a requirement. But since we could possibly use traditional

row, and column redundancy to achieve 100% perfect media and sell the memory as a DRAM at a higher cost, it is worthwhile to investigate including both column and row redundancy, with bad block marking. Bad block marking could be used in early production and, after the process matures, DRAM could be manufactured when the row and column redundancy is sufficient to fix all the faults. Since row and column redundancy are already included in the DRAM, we must examine the cost of including the non-volatile memory. The additional area requirement for BBM is only a small fraction of the area compared to what row, and column redundancy require, as shown above. Bad block marking can be included with traditional row and column redundancy at almost zero cost.

The analysis was done assuming that row and column redundancy is applied first and any 100% functional DRAM is removed and sold. The results are using the 1Gb DRAM used for the BBM analysis [27] and is using 128Kb of non-volatile memory. Then the remaining die are sold as file memory using BBM. Since a bad DRAM still has row and column redundancy, the number of defects that bad block marking needs to mark is decreased. Another strategy would be not to use the row and column redundancy as redundancy, but instead, to make those cells available at the pins as extra bits. Figure 3.8 below shows the results of combining BBM and traditional redundancy. The equation used to calculate the column and row redundancy is taken from Chapter 2. The equation required to plot the equivalent yield for bad block marking is Equation (3.10). The equation required to plot the equivalent yield using column row and bad block marking is replacing Y_{sc} with Y_{sccr} in Equation (3.10) where Y_{sccr} is from Equation (2.15).

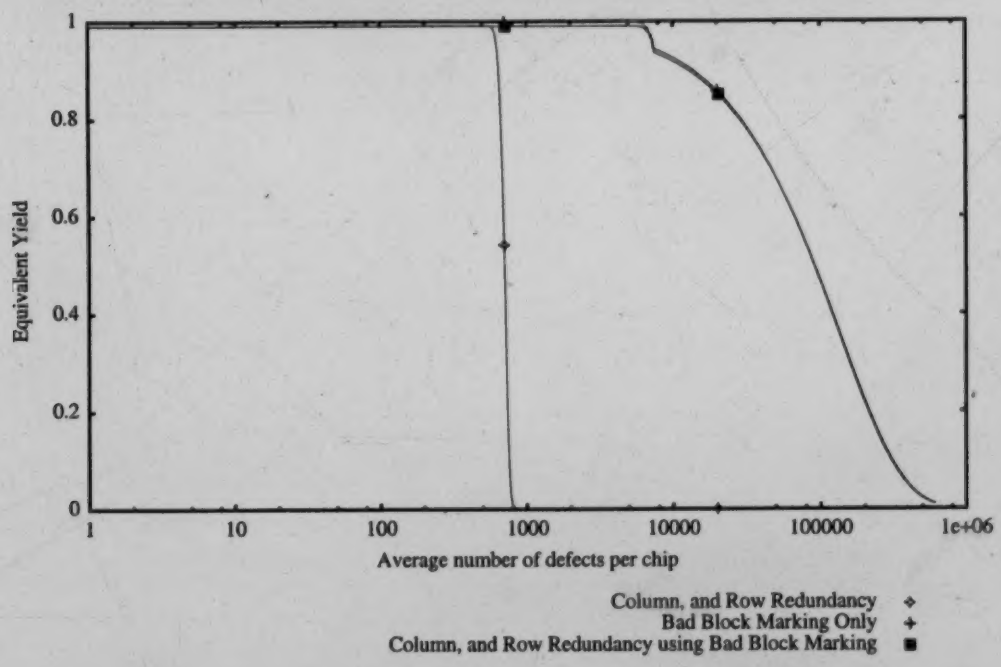


FIGURE 3.8. Equivalent Yield Using Bad Block Marking, Row and Column Redundancy

There are three lines in the figure, one is bad block marking, one is row, and column redundancy, and the last is bad block marking with row and column redundancy. It is hard to see in Figure 3.8 what the equivalent yields are when the average number of defects is lower. So the top portion of the figure is replotted below in Figure 3.9.

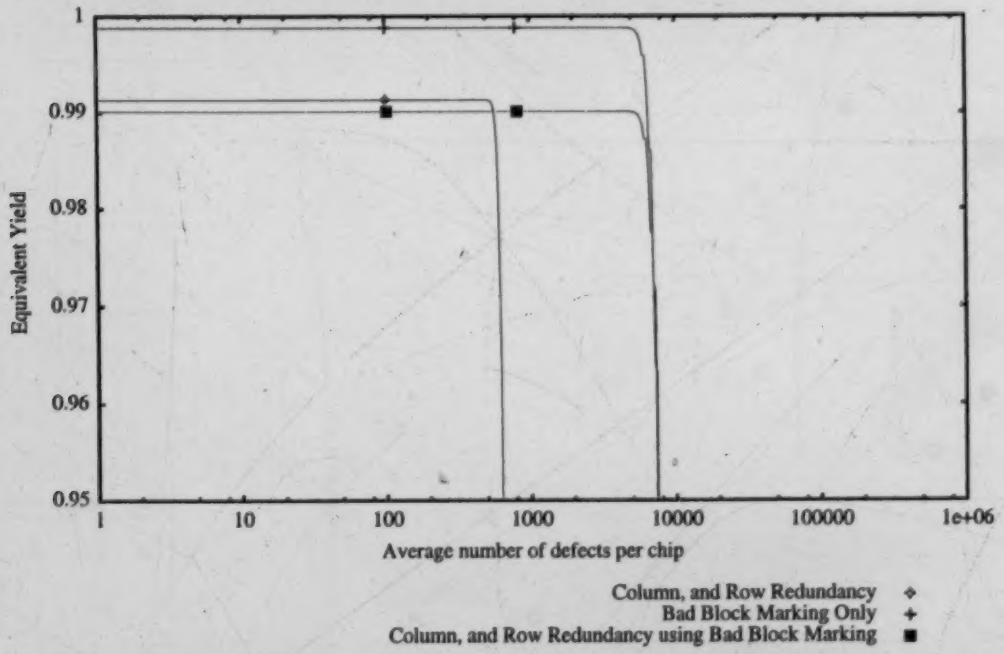


FIGURE 3.9. Equivalent Yield using Bad Block Marking, Row and Column Redundancy (Top portion)

We can see that using BBM with redundancy is worse than BBM only due to the extra overhead required by the row, and column redundancy. The benefit is the huge increase in the number of tolerable defects by a factor of over 100 times. The fifty percent equivalent yield points for the above figure are given in below in Table 3-2.

TABLE 3-2. 50% Equivalent Yield Points for BBM and Row, Column Redundancy

Redundancy Technique	Average Number of Defects
Row, and Column and Redundancy	705
Row, Column, and BBM	89559
BBM	90705

3.9 Conclusions

In the previous sections we have seen how the equivalent yield was effected when BBM was used. Now we can examine how BBM can be used during early DRAM

manufacturing. Using 128Kb of non-volatile storage on the chip works out to only about 0.12% of the area of the DRAM cell array. This is a relatively low area overhead and allows a high number of tolerable defects when compared to 0.82% area overhead for row and column redundancy.

If we assume that the manufacturer included both redundancy and BBM, then when the number of defects is high, as in initial production, the memory can be sold as file memory. After the process matures, the redundancy can be used to achieve perfect media and the memory can be sold as DRAM. The advantage to using BBM for the manufacturer is the gain of profit when defect levels are high. Figure 3.10 below shows at what average number of defects DRAM could be sold and when BBM could be sold. We can see that, when the defect levels are high, we are selling only BBM memory and, as the average number of defects decreases, we move to selling DRAM. The crossover occurs at approximately 705 defects. The profit from column and row redundancy is the equivalent yield and is taken from Chapter 2. The profit from bad block marking, column and row redundancy is the equivalent yield of bad block marking column and row redundancy minus the equivalent yield of column and row redundancy.

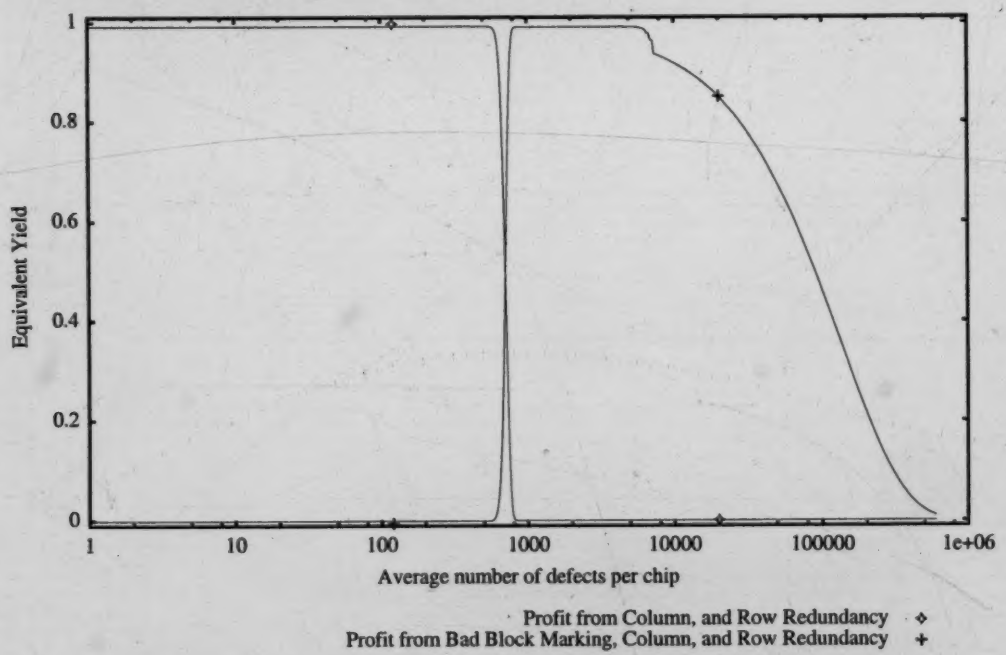


FIGURE 3.10. Profit Regions for Manufacturer for BBM and Redundancy

Chapter 4: Systems Measurements

In this chapter we measure and quantify the impact on throughput of the yield enhancement achievable using the techniques described in the previous chapters. The performance implications of performing the ECC correction off-chip are quantified. The performance implications of doing Bad Block Marking are also measured and normalized with respect to a system that has perfect media.

4.1 Hamming Code Performance

As mentioned earlier, error correction can be performed off-chip in order to reduce the on-chip ECC hardware. The correction could be performed instead in software on the CPU, reducing hardware cost but increasing the average access time. If we consider the total time needed to access a DRAM, for a very small number of defects it may be acceptable to do off-chip correction. The time required to perform off-chip correction is measured and compared to memory access alone since the hardware error detection and correction time is negligible compared to the memory access time [2].

4.1.1. Measuring the Performance Impact of Off-chip Correction

The performance of a computer performing the error correction in software was measured. A program was written that takes a Hamming code specification and generates the corresponding C code that will correct the ECC data. Once the C code was generated, a test program was written that determined the apparent access time for the memory and measured the time required to correct the data. The Hamming code was run on a Pentium II 333MHz machine, with both the level one and level two caches turned off, running Linux in single user mode. The Hamming code that was used was IBM's (137,128) code which contains 128 bits of data (i.e. 4 32-bit words). The average time to correct 128 bits of data was measured to be 30.29µs. This measurement was taken as the average of 400 data corrections of 128 data bits. Then, the performance degradation was measured.

The number of errors to be corrected was varied and the time required to perform the read and correction was compared to the time required to perform the read alone. An average number of failures per byte was calculated as the number of errors that occur divided by the total number of bytes read. It was assumed that the time required for the DRAM to generate the syndrome bits was negligible, and hence, the ECC DRAM performance at the pins is almost identical to the performance of a regular DRAM. This may not be true because an ECC DRAM may be slightly slower due to the time required to generate the syndrome bits. An ECC DRAM would be slightly slower than this due to the need to generate the syndrome bits and then to possibly correct the data.

4.1.2. Results of Off-chip Correction

Figure 4.1 shows the results of off-chip correction. The y axis is what we call the normalized performance, which is the time to access the memory (including error detection) over the total time required to access and correct the data. The normalized performance is the percentage performance degradation when correction of data is needed. The x axis shows the average number of errors that occur when reading a byte of data. It was shown in Chapter 2 in Table 2-7 that the 50% equivalent yield point occurs at an average of 1680 defects for the 1Gb DRAM This works out to an average number of errors per byte access of 1.25×10^{-5}. This average number of errors per byte corresponds to a normalized performance of 0.9963. Using this number as the normalized performance is pessimistic because we are assuming that all DRAM chips will have 1680 defects. In reality there will be many DRAM chips with fewer errors and thus a larger normalized performance.

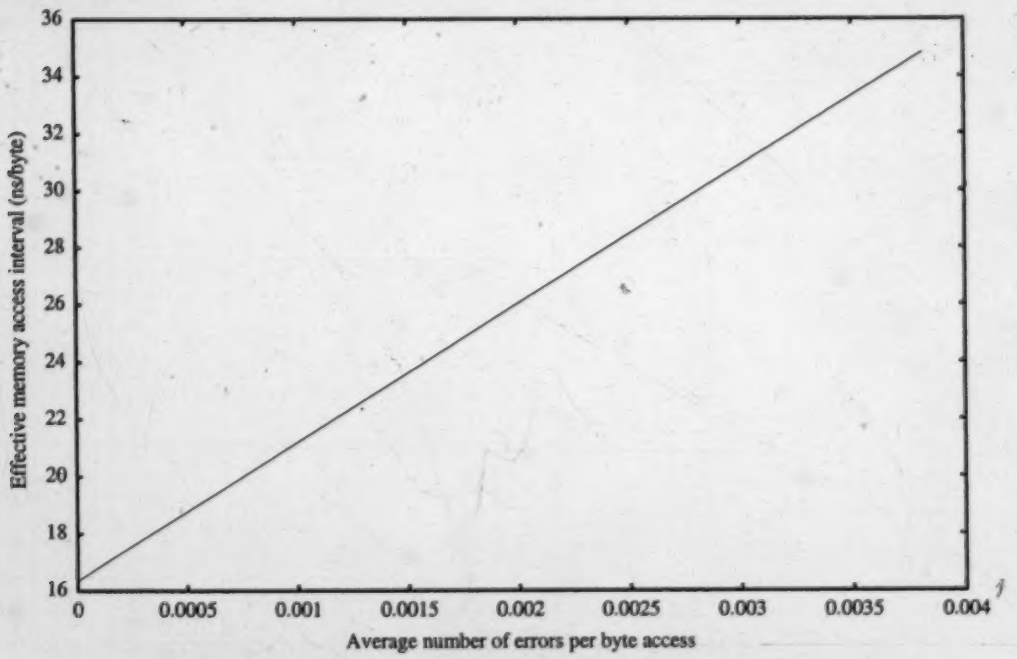


FIGURE 4.1. Performance Degradation With Off-Chip Error Correction

The performance depends on the memory access patterns, and on the location and frequency of the errors that need to be corrected. Since the x axis is the average number of errors per byte, every type of access pattern exists on the plot regardless of how many errors there are in the DRAM and the memory access patterns.

4.2 Bad Block Marking Performance

The bad block marking analysis in Chapter 3 examined how to mark portions of the memory as bad. Now file memory must be used in a system where the system must map around the bad portions of memory. This section measures this performance change in order to clarify where bad block marking might fit in the memory scheme. The performance penalty comes from requiring address translation. If bad block marking performs very well, then file memory may be able to replace DRAM at the cost of only a slightly slower computer. If it does not perform well enough, however, then an alternative application fit might be as a replacement for flash memory. Recall that we have relaxed the traditional constraint of having fast random access in order to target applications as listed in Table 1-2. It is assumed that bad block marking would be used for sequential accesses with a high bandwidth. It is important to remember that we are now at the systems level and we are now marking bytes as bad.

4.2.1. Bad Block Marking Performance Setup

A 333MHz Pentium II with 128Mb of DRAM with the level 1 and level 2 caches turned off was used to determine the performance of a simple bad block marking scheme. The caches were disabled using the BIOS. Linux was used in single user mode with a minimal amount of programs running. The test compared a memory copy, using a single bcopy, to a memory copy that uses a memory marking list and multiple bcopies. Here the time was measured as the average time of many access. For example the time required to perform a bad block marking copy is measured as the average of ten bad block marking copy function calls. Where the bad block marking copy function call execution time is measured as the average of ten singular bad block marking copies. This gives us the average execution time for the bad block marking copy. This time is then measured six times with each measurement being plotted. Each time was plotted six times to see how much variance there was in the performance.

4.2.2. Bad Block Marking Scheme

The data structure used is a two-dimensional array that contains the beginning address (specified as an offset from the memory base address) and the amount of memory after that address that is functional. A bad block copy was then written that uses the bad block marking list and multiple bcopies to perform the BBM memory copy. This experiment is consistent with a sequential access pattern. One thing to note is that bytes are marked bad at the software level while blocks of bits are marked at the hardware level. Words could have been marked just as easily at the software level. This was done only to make the software required for bad block marking as fast as possible. We could mark bits as bad at the software level too by keeping a mask for each byte that contains single or multiple bad bits. This mask and bit shifting could then be used to regain all of the good bits. There are three potential problems with doing this. The first is that it would slow down the access considerably due to the software overhead required for masking and shifting. The second is that storing the location of the bad (or good) bits would require additional memory. For

example, if one bit is bad, the location can be stored in as little as three bits which costs three bits to salvage 7 bits, which is a gain of a little over 60%. If there are 2 bad bits then we could code each bit position separately with 6 bits (3 bits for each error), however, this is a zero percent gain and a speed loss. We do need to acknowledge, however, that 2 errors could be marked using 5 bits which saves one bit or about 12% if the errors are coded together into the same bits (recall $\binom{8}{2} = 28 < 32 = 2^5$). The third problem is the percentage of good bits that get recovered over the total number of bits already available. For example, if there are 2000 single cell failures in a 256Mbit DRAM, then we are losing 14000 good bits by marking the entire byte as bad, which is only 0.005215 percent of the capacity.

4.2.3. Results on a 333MHz PII With Disabled Cache

Figure 4.2 shows the access time for a memory copy and a memory copy that uses the bad block marking technique. The horizontally plotted points represent the amount of time required to copy a 2MB test file from memory to memory. This is used as a baseline to compare how much slower the bad block marking technique is. We can see that as the number of separate blocks marked increases, the time required to mark them also increases. It should be noted that the x-axis is not the average number of defects, as in other figures. Instead it is the number of separate blocks that are marked bad. An analysis using the average number of defects is given later. For certain values of the number of blocks marked bad, the bad block marking scheme performs surprisingly well. The reason for this has not been investigated, but I believe that this is due to page or address alignment effects that can achieve better results when a bcopy is called. One thing to note is that these are average run times and we can see that when the number of separate blocks marked is low, the performance of both is almost identical. This means that when we take the average run times, it is possible for the BBM to appear to be better than the DRAM speed and we will get a ratio larger than the ratio when the two are compared. The

difference in average run times for different numbers of marked blocks is due to the operating system.

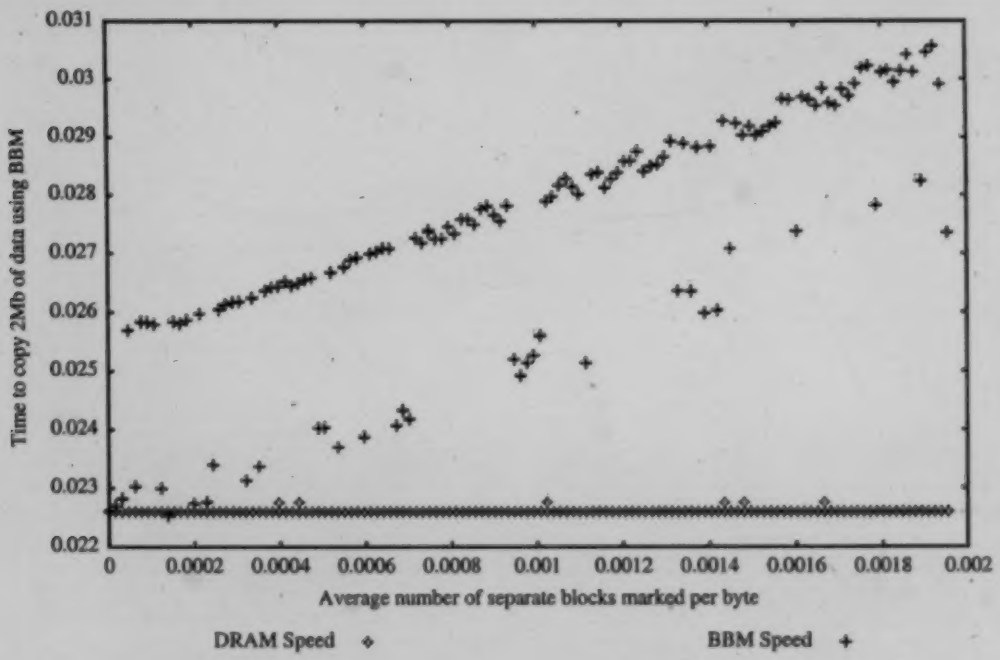


FIGURE 4.2. Access Time for Normal DRAM and for a Simple Bad Block Marking List

Figure 4.3 shows the ratio of the two time measures. This result shows us how fast the BBM scheme can perform when compared to regular DRAM.

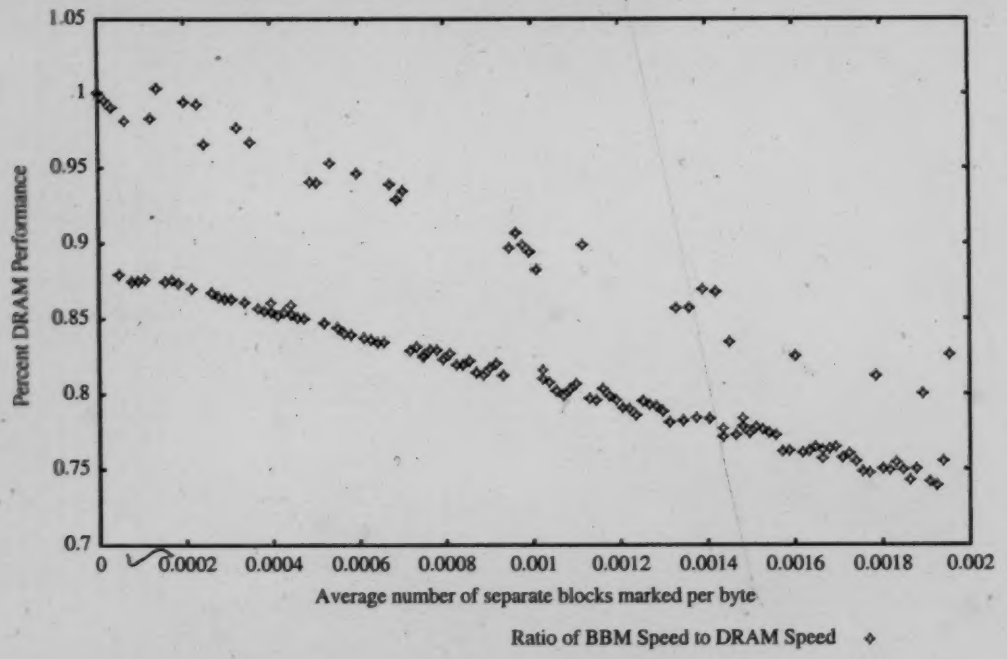


FIGURE 4.3. Performance of Bad Block Marking on a 333MHz PII PC

4.2.4. Results Using Average Number of Defects

As mentioned earlier, we can make comparisons using the average number of defects and not the number of blocks marked as bad. We can then replot Figure 4.3 with the average number of defects along the x-axis. Figure 4.4 shows the results of using the average number of defects. We can see that the performance is more linear than shown above. This is because, as the average number of defects increases linearly, the number of pages required to mark the errors increases less than linearly. At an average of 2048, 4096, and 6144 defects, the number of separate pages required to mark the DRAM is 1556, 2276, and 2432, respectively. These numbers assume a Poisson distribution of errors in the pages. In other words, as the average number of defects increases, the number of pages that need to be marked does not increase at the same rate. It was assumed that there were

8Kb of non-volatile memory available for bad block marking. The optimal block size was chosen for each average number of defects, as shown in Section 3.2.

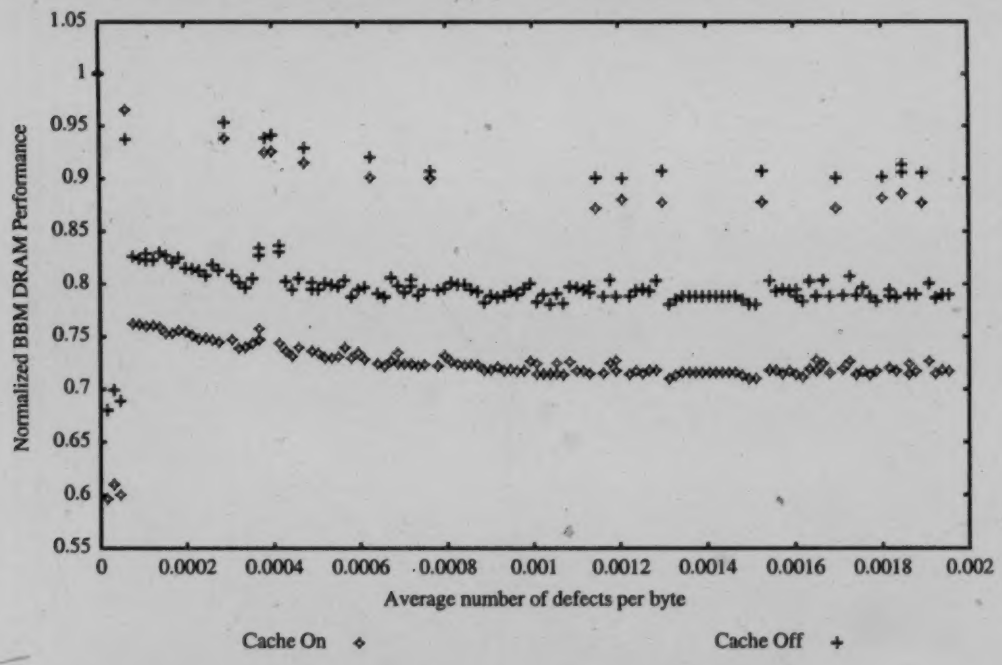


FIGURE 4.4. Real Performance of a Bad Block Marking Scheme

We can see that the performance is about 80% of that of conventional DRAM, with a very slow degradation in performance as the number of blocks that are marked bad is increased. There are two sets of data points in the above figure. One set represents the performance with the cache enabled and the other represents the performance with no cache. The bad block marking technique performs worse when the cache is on. The overall performance of bad block memory is roughly 75% that of regular DRAM when the cache is turned on, and it is approximately 80% that of regular DRAM when the cache is turned off. When the cache is turned on we achieve lower performance because the amount of data that we are transferring is much larger than the cache size. When the cache is turned on, a larger amount of time is spent transferring the data into the cache when the cache cannot make efficient use of it. This is expected because the data access patterns do not exhibit spatial or temporal locality. A possible reason that the performance is worse when the average number of defects is low is that a list structure is used and the block size is quite small. Recall that for each average number of defects there are six individual points plotted. All six points lie on top of each other with a variance of less than 0.5%.

4.2.5. Bad Block Marking Random Access Performance

Although bad block marking is not intended to be used for random access patterns, it can still be done and the performance of random memory accesses using bad block

marking can be measured. The linear data structure that is used for the previous performance analysis is good for sequential accesses but is not very good for random access. A binary tree representation of the bad block marking list was, therefore, created. This binary tree is then searched to achieve an average seek time for any random byte that increases with the logarithm of the size of the data structure. Figure 4.5 below shows the performance of the BBM list structure and of the binary search structure. It can be seen that the binary search performs better than the list structure which is expected. It is hard to see on the graph but the list structure performs better when there are very few defects. This is because the overhead per iteration is less for the list search. The steps that can be seen on the binary search tree structure are from the depth of the tree changing when the average number of defects becomes sufficiently large. The time required for the address to be found, for the worst case using a binary tree, was measured as the average of 800 address lookups. The program was run on a PII333Mhz with level one and level two caches turned off. The list structure was measured in the same way on the same machine. The time required to perform only the memory copy (without the use of either structure) was also measured and this is what the two schemes were compared against. The Y axis gives the ratio of the time required with no schemes to the time required for the technique being measured. The performance measured is the worst case. The time required for the binary search to find the byte for the worse case is 164ms for an average of 512 defects, and 213ms for an average of 4096 defects.

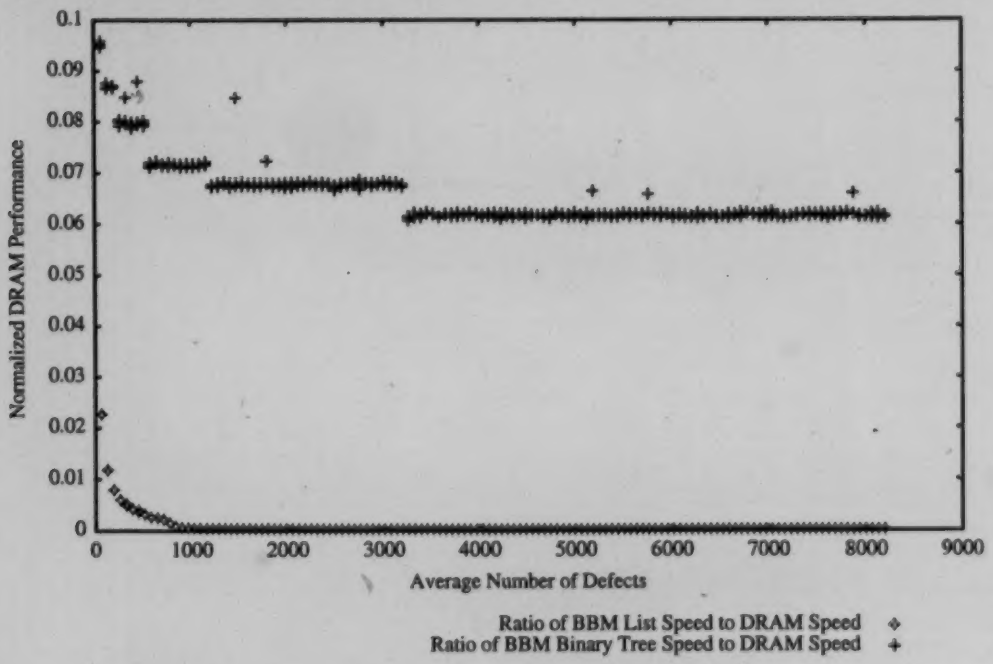


FIGURE 4.5. Random Access Performance Using BBM

4.3 Summary

The throughput performance of doing correction off-chip ECC correction was measured to be 99.96% of the performance of regular DRAM at a 50% equivalent yield for a 1Gb DRAM [27]. BBM techniques and marking structures were described and examined. The two major structures presented are a bitmap marking scheme, and a list marking scheme. The software performance of bad block marking was measured to be approximately 75% that of regular DRAM. The proposed software support for file memories does not unduly impact memory system performance.

Chapter 5: ECC Implementation and Area

This chapter examines the ECC implementation overhead by producing physical IC layouts of multiple implementation. A C program was written that takes a Hamming code specification in the form of a text file and converts the Hamming code into a VHDL circuit description at the gate level. The generated VHDL code represents the hardware required to generate the check bits as well as correct the data bits that were read from memory. The VHDL code was then compiled using the Synopsys Design Analyzer and was targeted towards CMC's wcell standard cell library using a 0.35µm CMOS technology. The number of standard cells required and the cell area were then determined.

To verify that the area of the standard cells is roughly the area required for the entire circuit, the cell layout was also placed and routed. The results of the place and route are shown below in Figure 5.1 for a (22,16) Hamming code. This (22,16) code uses the Hamming code given in Appendix B.1. Alternate rows of the layout were rotated by 180 degrees so that the cells could abut and use the same power rails. The place and route was successful with this code. A larger placed and routed (137,128) code is shown in Appendix A.1 which uses the Hamming code given in Appendix B.5. When Synopsys was used to compile the VHDL, no timing constraints were set. This means that as the codes become larger they become much slower because Synopsys will choose the smallest and slowest standard cells in order to minimize area. This is not very realistic as there will always be a timing requirement for the circuits, but our results should still give us an approximate area for the circuit. This is the reason why the timing performance of the final synthesized circuits was not measured. Timing constraints were not set because we have no specification or requirements for timing. If a timing constraint were set then the circuit could be synthesized to meet this performance requirement. We did, however, have the timing results from Synopsys that gave the worst case delay of the gates without the wiring.

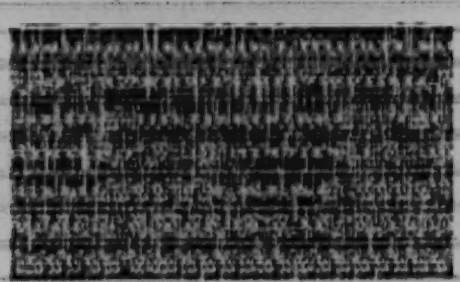


FIGURE 5.1. (22,16) Hamming Code Layout

When place and route was tried with a larger (523,512) code, the router could not route the design due to insufficient routing space. The cells needed to be spaced out more left to right with an 85% area usage (the cells took up 85% of the possible area along the

placement rows). The rows need to be spaced vertically with a 2.8μm space between each row. For that layout the alternate rows were not rotated 180 degrees because they did not abut. It should be noted that using place and route to build the ECC hardware will yield a pessimistic result as the layout could be made smaller by creating datapath cells that are optimized for ECC circuits. Alternatively, the whole ECC circuit could be laid out by hand and the area could be reduced further. Figure 5.2 shows how the number of required standard cells changes with the different code sizes.

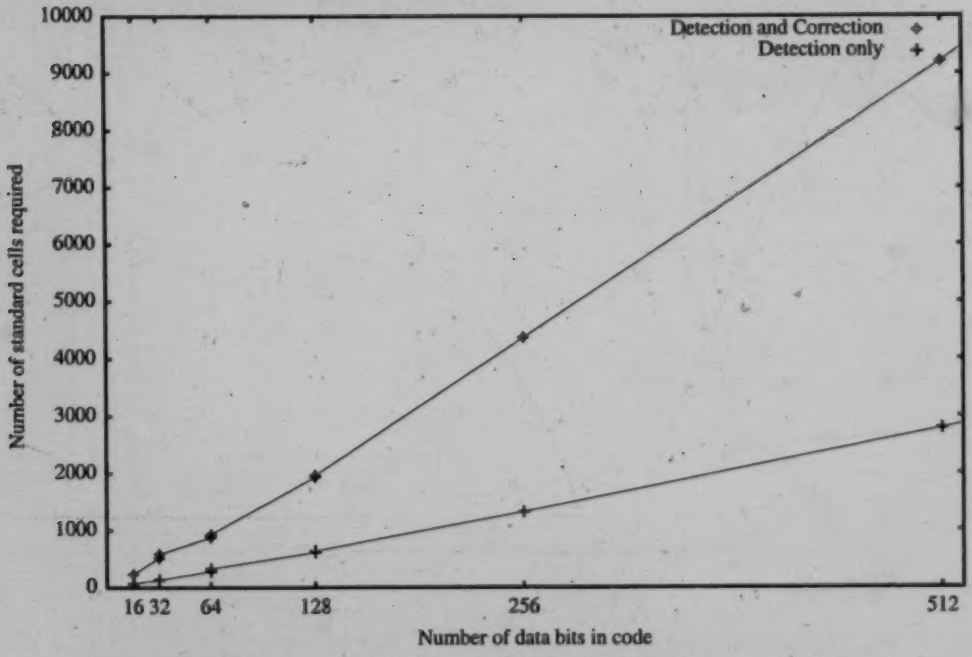


FIGURE 5.2. Circuit Size for Detection, and Detection and Correction

The above figures use the number of standard cells as a measurement of the size of the ECC circuitry. Since not all standard cells will be the same size, we should really use silicon area to compare the different circuits. Figure 5.3 shows how the area changes when different code sizes are used. There are two different areas shown for the full correction and detection. One is the area required for the standard cells and the second is the area required for the full placed and routed circuit. Once a placement was achieved that would route no further, placement optimizations were made. We can see that the figure looks almost identical to Figure 5.2 with the exception of the y-axis scale. This supports using

the number of cells as a comparison of relative size because the area of the different types of standard cells used does not vary much.

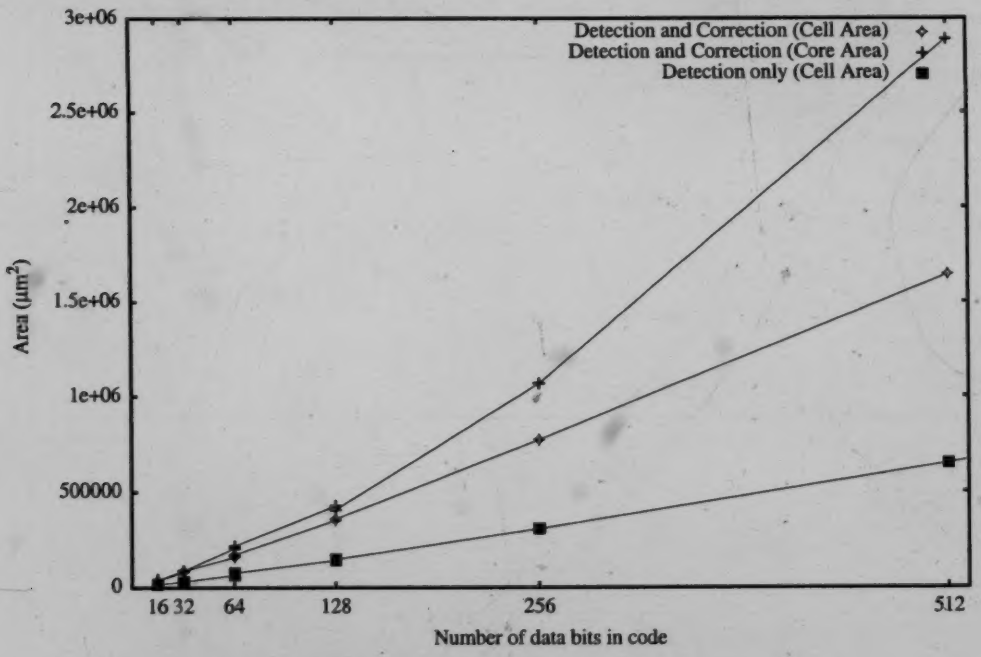


FIGURE 5.3. Comparison of Area of Standard Cells for Detection, and Detection/Correction

The use of different codes was also analyzed. Codes were randomly chosen out of the entire pool of possible codes. These codes where synthesized and compared. The results showed that all of the random codes were about 10% greater in area and 10% slower in performance than the IBM code (using only the standard cell area and delay generated by Synopsys). This was also expected because the random codes use more bits for the coding than were used in the IBM code and thus result in larger and slower circuits. This implies that the order of the code matrix rows chosen does not matter, but the choice of code matrix does make a difference. These results confirm the area advantages of doing the correction off chip as proposed in Chapter 2.

5.1 ECC Circuit Area Overhead

Up to now figures that show the equivalent yield with the use of ECC only account for the area required by the extra cells and do not account for the area required by the ECC correction/detection circuity. We can use the area requirements for ECC codes from the above analysis in this section to calculate the area requirement if we used one of the 1Gbit DRAM chips from Table 1-3. The smallest 1Gb is [36] with a size of 581.77mm^2 using a 0.15μm process. The area requirement for a (266,256) ECC using a 0.35μm process was

1.067mm^2. Converting this area to the 0.15µm process we get an area of 0.196mm^2. This works out to an area overhead of 0.0337% for the ECC circuitry, and an area overhead of 41943040 bits. This corresponds to an increase of 3.9% in the number of bits. The low area overhead for the ECC circuity means that the analysis where just the change in memory area is used as a measure of area are valid. It does, however, also mean that the analysis of saving die area by doing correction off-chip is not useful because saving 75% of 0.0337% is a savings of 0.0253% which is probably not worth the loss in speed performance and increase in system integration effort. It should be noted that the area requirement for wiring from the array to the ECC circuitry was not included in this overhead as they should abut.

5.2 Summary

The area required by ECC circuitry was determined by placing and routing several different ECC codes. The area savings of 75% if detection only is performed was verified. It was also determined that this savings of 75% is actually only a savings of 0.135% over the die because the DRAM cell area dominates and the major overhead when using ECC is the data bits required to store the check bits.

Chapter 6: Conclusion

6.1 Summary of Results

Several techniques have been presented for decreasing the cost of memory (by increasing the yield) that can tolerate higher manufacturing defect rates than conventional DRAM. The effective yield measure was used as a more meaningful comparison method for circuits that use fault tolerance techniques. The presented techniques used for tolerating faults were Hamming ECC, column and row redundancy, and bad block marking (BBM). Combinations of these techniques were also presented. The average number of tolerable defects were 28, 705, 1680, 10866, 54751, 89559, and 90705 for column only, row/column, ECC only, ECC and column, ECC row/column redundancy, bad block marking with row/column redundancy, and bad block marking only, respectively, at a 50% equivalent yield. For these bad block results, a non-volatile memory size of 128Kb was used which requires an area overhead of 0.12%. Traditional row and column redundancy requires an overhead of 0.82%.

The use of techniques such as ECC can correct hard errors, and in this way, increase the yield; however, such techniques can also be used to correct soft errors and create a robust memory system from underlying memory with perhaps less than industry standard reliability (100FIT). The work done by IBM using ECC and redundancy was extended to a modern 1Gb DRAM and a synergistic fault tolerance effect was still observed. Several VHDL models using different Hamming Codes were created in order to verify the area cost of a placed and routed circuit. The area cost of the full Hamming code with error detection and correction, was compared to the area of a circuit that provides only detection. The performance penalty of having the CPU do the error correction in software was then examined. The resulting performance depends on the number of errors, location of the errors, and the memory access patterns. The results did show that the performance penalty is not very significant for a low number of defects per access. The area overhead for the ECC circuitry was extended to a modern 1Gb DRAM [36] and it was determined that the area overhead of the circuitry would be 0.0337% with an increase in area due to the ECC check bits of 3.9% using a (266,256) Hamming code. This means that even though we can save 75% of the ECC circuitry area by removing the correction circuitry we would probably not, due to it actually saving around 0.0253% of the total area.

Several aspects of BBM were investigated including the performance of BBM, the optimal block size of BBM, and the achievable equivalent yield. The speed performance of using BBM was determined to be approximately 75% that of regular DRAM assuming sequential data access patterns. The optimal size for BBM is a moving target and changes with the average number of defects. As it is not reasonable to frequently change the amount of non-volatile memory on the DRAM, the amount of non-volatile memory can be set so that a given average number of defects can be tolerated. Given a fixed size of non-volatile memory, the way in which the BBM information should be stored in the given non-volatile memory was analyzed. A few different possibilities were presented. The first was to use a list that represents the address of the bad blocks, the second was to use a

bitmap representation, and the third was to use a list to mark the good blocks (which is appropriate if the average number of defects is high). Specific techniques can be developed for specific manufacturing defects. For example, a way to mark bitlines as bad was designed. The technique developed uses approximately 350 times less non-volatile memory than marking bits. BBM was also compared to traditional redundancy techniques. Using 2Kb of non-volatile memory for the BBM, a memory could tolerate twice as many errors as row, and column redundancy with approximately 430 times less area.

While these techniques may interfere to some extent with conventional random accesses to memory, file-oriented applications of memory can more easily accept these limitations. These limitations are: extended access time, non-uniform access times and discontinuous regions of good memory. As an example of non-uniform access time, significant area overhead can be saved if the memory chips only detect errors in the codeword and correction is done off-chip. The fault tolerance techniques described could hasten the economic crossover for higher density DRAM generations (e.g. 1Gb and 4Gb).

6.2 Future Work

The ECC investigated was the Hamming code. Such a code is useful when the location of the error is not known and when hard or soft defects are introduced after being tested. It was assumed, for this work, that the hard defects do not change over the life of the DRAM, as is usually the case. Investigation of slower but more powerful ECC coding techniques, needs to be done because they fit the file memory model better, and they may result in a higher equivalent yield. Fault modeling for more current and future DRAM is necessary so that appropriate BBM structures can be designed and built. Doing the fault modeling will allow better use of the available non-volatile memory which will increase the effectiveness of BBM. The performance of BBM was determined with a fixed amount of non-volatile memory. The amount of non-volatile memory was fixed at 8Kbits. The performance for different sizes of non-volatile memory should be done and be done for different systems. Implementing BBM for an embedded systems and measuring the performance. In this work it is assumed that we are starting with a standard DRAM chip which has the number of bitlines and the wordlines as a power of two. Since we are investigating sequential-mode block access, there is no reason that the blocks must be a power of two in size. Therefore, investigation of block sizes and architectures of irregular sizes is an appropriate subject for future research.

Bibliography

[1] C.H. Stapper, "Synergistic Fault-Tolerance for Memory Chips", IEEE Transactions on Computers, Vol. 41, No. 9, Sept. 1992, pp. 1078-1087.

[2] H.L. Kalter, et al., "A 50-ns 16Mb DRAM with a 10-ns Data Rate and On-Chip ECC", IEEE Journal of Solid-State Circuits, Vol. 25, No. 5, Oct. 1990, pp. 1118-1128.

[3] Takaaki Nozaki, et al.,"A 1-Mb EEPROM with MONOS Memory Cell for Semiconductor Disk Application", IEEE Journal of Solid-State Circuits, Vol. 26, NO. 4, April 1991, pp. 497-501.

[4] P. Gillingham, "A Sense and Restore Technique for Multilevel DRAM," IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 43, No. 7, pp. 483-486, July 1996.

[5] T. Okuda et al., "A Four-Level Storage 4-Gb DRAM," IEEE J. Solid-State Circuits, Vol. 32, No. 11, pp. 1743-1747, Nov. 1997.

[6] Steve Mann, "Wearable Computing: A First Step Toward Personal Imaging," IEEE Computer, Vol. 30, No. 2, pp. 25-32, February 1997.

[7] G. Birk, D. G. Elliott, and B. F. Cockburn, "A Comparative Study of Four Multilevel DRAMs," 1999 IEEE International Workshop on Memory Technology, Design and Testing, San Jose, CA, U.S.A., Aug. 9-10, 1999, pp. 102-109.

[8] Anthony Cataldo, "Toshiba readies double-data-rate 256-Mbit DRAMs," EETimes, July 18, 2000, http://www.eetimes.com/story/OEG20000718S0037.

[9] Raul Cernea, et al., "A 34Mb 3.3V Serial Flash EEPROM for Solid-State Disk Applications", IEEE International Solid-State Circuits Conference, 1995, pp. 126-127.

[10] Goro Kitsukawa, et al., "256-Mb DRAM Circuit Technologies for File Applications", IEEE Journal of Solid-State circuits, VOL.28, NO 11, November 1993,pp. 1105-1113.

[11] J.A. Fifield and C.H. Stapper, "High-Speed On-Chip ECC for Synergistic Fault-Tolerant Memory Chip", IEEE Journal of Solid-State Circuits, Vol. 26, No. 10, pp. 1449-1452, Oct. 1991.

[12] J.A. Fifield, "A High-Speed On-Chip ECC System Using Modified Hamming Code", in Sixteenth Euro. Solid-State Circuits Conf. Proc., Sept. 1990, pp. 265-268.

[13] Fazil I. Osman, "Error-Correction Technique for Random-Access Memories", IEEE Journal of Solid-State circuits, Vol. SC-17,NO. 5, October 1982, pp. 877-881.

[14] Mikio Asakura, et al.,"An Experimental 1-Mbit Cache DRAM with ECC", IEEE Journal of Solid-State circuits, Vol. 25, No 1, February 1990,pp. 5-10.

[15] Kazutami Arimoto, et al., "A Speed-Enhanced DRAM Array Architecture with Embedded ECC", IEEE Journal of Solid-State circuits, Vol. 25, No 1, February 1990, pp. 11-17.

[16] Toshio Yamada, et al., "A 4-Mbit DRAM with 16-bit Concurrent ECC", IEEE Journal of Solid-State circuits, Vol. 23, No 1. February 1988,pp. 20-25.

[17] Toshiaki Kirihata, et al., "Fault-Tolerant Designs for 256 Mb DRAM", IEEE Journal of Solid-State circuits, Vol. 31, No 4, April 1996,pp. 558-566.

[18] Masashi Horiguchi, et al., "A Flexible Redundancy Technique for High-Density DRAM's, IEEE Journal of Solid-State circuits, Vol. 26, No. 1, January 1991,pp. 12-17.

[19] Amitai, Z.; Rajpal, S.; Sachs, N.,"Improving your DRAM reliability by two different VLSI solutions", Southcon/84. Electronics Show and Convention, pp. 9/2/1-14.

[20] Koichi Sakurai, et al., "A defect-situation forecasting technology to optimize future DRAM-redundancy design", 1997 IEEE International Symposium on Semiconductor Manufacturing Conference Proceedings, pp. E43-6.

[21] B. Prince, "Semiconductor Memories", John Wiley & Sons, 2nd Edition, 1991.

[22] C.H. Stapper; A.N. McLaren; M. Dreckmann, "Yield Model For Productivity Optimization of VLSI Memory Chips With Redundancy and Partially Good Product", IBM J. R.&D. Vol. 24, No. 3, pp. 398-409, May 1980.

[23] Heller et al., "Cascode Voltage Switch Logic", ISSCC Dig. Tech. Papers 1984, p. 16-17.

[24] D. H. K. Hoe, et al., "Cell and Circuit Design for Single-Poly EPROM", IEEE Journal of Solid-State Circuits, Vol. 24, No. 4, Aug. 1989, pp. 1153-1157.

[25] M.Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes", IBM J. R.&D., July 1970, pp. 395-401.

[26] D.P. Siewiorek, et al., "Reliable Computer Systems", A. K. Peters, Natick, Massachusetts, 3rd Edition 1998, pp. 771-781.

[27] Kyuchan Lee, et al., "A 1 Gbit Synchronous Dynamic Random Access Memory with an Independent Subarray-Controlled Scheme and a Hierarchical Decoding Scheme", IEEE Journal of Solid-State Circuits, Vol. 33, No. 5, May. 1998, pp. 779-786.

[28] Kirihata et al., "A 390mm^2 16 Bank 1Gb DDR SDRAM with Hybrid Bitline Architecture", ISSCC Dig. Tech. Papers 1999, p. 422-423.

[29] Takai et al., "A 250Bb/s/pin 1Gb Double Data Rate SDRAM with a Bi-Directional Delay and an Inter-Bank Shared Redundancy Scheme", ISSCC Dig. Tech. Papers 1999, p. 418-419.

[30] Kimura et al., "64Mb 6.8ns Random Row Access DRAM Macro for ASICs", ISSCC Dig. Tech. Papers 1999, p. 416-417.

[31] Yoon et al., "A 2.5V 333Mb/s/pin 1Gb Double Data Rate SDRAM", ISSCC Dig. Tech. Papers 1999, p. 412-413.

[32] Eto et al., "A 1Gb SDRAM with Ground Level Precharged Bitline and Non-Boosted 2.1V Word Line", ISSCC Dig. Tech. Papers 1998, p. 82-83.

[33] Hasegawa et al., "A 256Mb SDRAM with Subthreshold Leakage Current Suppression", ISSCC Dig. Tech. Papers 1998, p. 80-81.

[34] T. Kirihata et al., "A 220mm^2 4 and 8 Bank 256Mb SDRAM with Single-Sided Stitched WL Architecture", ISSCC Dig. Tech. Papers 1998, p. 78-79.

[35] Yoo et al., "A 32-Bank 1Gb DRAM with 1GB/s Bandwidth", ISSCC Dig. Tech. Papers 1996, p. 378-379.

[36] Nitta et al., "A 1.6GB/s Data-Rate 1Gb Synchronous DRAM with Hierarchical Square-Shaped Memory Block and Distributed Bank Architecture", ISSCC Dig. Tech. Papers 1996, p. 376-377.

[37] Saeki et al., "A 2.5ns Clock Access 250MHz 256Mb SDRAM with a Synchronous Mirror Delay", ISSCC Dig. Tech. Papers 1996, p. 374-375.

[38] Sugibayashi et al., "A 1Gb DRAM for File Applications", ISSCC Dig. Tech. Papers 1995, p. 254-255.

[39] Horiguchi et al., "An Experimental 220Mhz 1Gb DRAM", ISSCC Dig. Tech. Papers 1995, p. 252-253.

[40] Yoo et al., "A 150MHz 8-Banks 256M Synchronous DRAM with Wave Pipelining Methods", ISSCC Dig. Tech. Papers 1995, p. 250-251.

[41] Nakamura et al., "A 29ns 64Mb DRAM with Hierarchical Array Architecture", ISSCC Dig. Tech. Papers 1995, p. 246-247.

[42] Tanoi et al., "A 32-Bank 256Mb DRAM with Cache and TAG", ISSCC Dig. Tech. Papers 1994, p. 144-145.

[43] Kotani et al., "A 256Mb DRAM with 100Mhz Serial I/O Ports for Storage of Moving Pictures", ISSCC Dig. Tech. Papers 1994, p. 142-143.

[44] Asakura et al., "A 34ns 256Mb DRAM with Boosted Sense-Ground Scheme", ISSCC Dig. Tech. Papers 1994, p. 140-141.

[45] Sugibayashi et al., "A 30ns 256Mb DRAM with Multi-Divided Array Structure", ISSCC Dig. Tech. Papers 1993, p. 50-51.

[46] Kitsukawa et al., "256Mb DRAM technologies for File Applications", ISSCC Dig. Tech. Papers 1993, p. 48-49.

[47] Hasegawa et al., "An Experimental DRAM with a NAND-Structured Cell", ISSCC Dig. Tech. Papers 1993, p. 46-47.

[48] Koike et al., "A 30ns 64Mb DRAM with Built-in Self-Test and Repair Function", ISSCC Dig. Tech. Papers 1992, p. 150-151.

[49] Oowaki et al., "A 33ns 64Mb DRAM", ISSCC Dig. Tech. Papers 1991, p. 114-115.

[50] Taguchi et al., "A 40ns 64Mb DRAM with Current-Sensing Data-Bus Amplifier", ISSCC Dig. Tech. Papers 1991, p. 112-113.

[51] Mori et al., "A 45ns 64Mb DRAM with a Merged Match-line Test Architecture", ISSCC Dig. Tech. Papers 1991, p. 110-111.

[52] Yamada et al., "A 64Mb DRAM with Meshed Power Line and Distributed Sense-Amplifier Driver", ISSCC Dig. Tech. Papers 1991, p. 108-109.

[53] Kimura et al., "A Block-Oriented RAM with Half-Sized DRAM Cell and Quasi-Folded Data-Line Architecture", ISSCC Dig. Tech. Papers 1991, p. 106-107.

[54] Kalter et al., "A 50ns 16Mb DRAM with a 10ns Data Rate", ISSCC Dig. Tech. Papers 1990, p. 232-233.

[55] Konishi et al., "A 38ns 4Mb DRAM with a Battery Back-up (BBU) Mode", ISSCC Dig. Tech. Papers 1990, p. 230-231.

[56] Fischer et al., "A 600Mhz IA-32-Microprocessor with Enhanced Data Streaming for Graphics and Video", ISSCC Dig. Tech. Papers 1999, p. 98-99.

[57] Alvarez et al., "450MHz PowerPC Microprocessor with Enhanced Instruction Set and Copper Interconnect", ISSCC Dig. Tech. Papers 1999, p. 96-97.

[58] Hesley et al., "A 7th-Generation x86 Microprocessor", ISSCC Dig. Tech. Papers 1999, p. 92-93.

[59] Northrop et al., "600MHz G5 s/390 Microprocessor", ISSCC Dig. Tech. Papers 1999, p. 88-89.

[60] Philip Barnes, "A 500Mhz 64b RISC CPU with 1.5MB On-Ship Cache", ISSCC Dig. Tech. Papers 1999, p. 86-87.

[61] R. Khanna et al., "A 0.25mmx86 Microprocessor with a 100MHz Socket 7 Interface", ISSCC Dig. Tech. Papers 1998, p. 242-243.

[62] N. Rohrer et al., "A 480MHz RISC Microprocessor in a 0.12μm L_{eff} CMOS Technology with Copper Interconnects", ISSCC Dig. Tech. Papers 1998, p. 240-241.

[63] J. Schutz et al., "A 450MHz IA32 P6 Family Microprocessor", ISSCC Dig. Tech. Papers 1998, p. 236-237.

[64] S. Storino et al., "A Commercial Multi-threaded RISC Processor", ISSCC Dig. Tech. Papers 1998, p. 234-235.

[65] Gronowski et al., "A 433MHz 64b Quad-Issue RISC Microprocessor", ISSCC Dig. Tech. Papers 1996, p. 222-223.

[66] Norrod et al., "A Multimedia-Enhanced x86 Processor", ISSCC Dig. Tech. Papers 1996, p. 220-221.

[67] Sanchez et al., "A 200Mhz 2.5V 4W Superscalar RISC Microprocessor", ISSCC Dig. Tech. Papers 1996, p. 218-219.

[68] Montanaro et al., "A 160MHz 32b 0.5W CMOS RISC Microprocessor", ISSCC Dig. Tech. Papers 1996, p. 214-215.

[69] Lotz et al., "A Quad-Issue Out-of-Order RISC CPU", ISSCC Dig. Tech. Papers 1996, p. 210-211.

[70] Bowhill et al., "A 300MHz 64b Quad-Issue CMOS RISC Microprocessor", ISSCC Dig. Tech. Papers 1995, p. 182-183.

[71] Pham et al., "A 1.2W 66MHz Superscalar RISC Microprocessor for Set-Tops, Video Games, and PDAs", ISSCC Dig. Tech. Papers 1995, p. 180-181.

[72] A. Charnas et al., "A 64b Microprocessor with Multimedia Support", ISSCC Dig. Tech. Papers 1995, p. 178-179.

[73] Bearden et al., "A 133MHz 64b Four-Issue CMOS Microprocessor", ISSCC Dig. Tech. Papers 1995, p. 174-175.

[74] Draper et al., "A 93MHz, X86 Microprocessor with On-Chip L2 Cache Controller", ISSCC Dig. Tech. Papers 1995, p. 172-173.

[75] Shen et al., "A 64b 4-Issue Out-of-Order Execution RISC Processor", ISSCC Dig. Tech. Papers 1995, p. 170-171.

[76] Saito et al., "A 1.71M-Transistor CMOS CPU Chip with a Testable Cache Architecture", ISSCC Dig. Tech. Papers 1993, p. 86-87.

[77] Abu-Nofal et al., "A Three-Million-Transistor Microprocessor", ISSCC Dig. Tech. Papers 1992, p. 108-109.

[78] Dobberpuhl et al., "A 200MHz 64b Dual-Issue CMOS Microprocessor", ISSCC Dig. Tech. Papers 1992, p. 106-107.

[79] Badeau et al., "A 100MHz Macropipelined CISC CMOS Microprocessor", ISSCC Dig. Tech. Papers 1992, p. 104-105.

[80] Joe Schutz et al., "A CMOS 100MHz Microprocessor", ISSCC Dig. Tech. Papers 1991, p. 90-91.

[81] Tanksalvala et al., "A 90MHz CMOS RISC CPU Designed for Sustained Performance", ISSCC Dig. Tech. Papers 1990, p. 52-53.

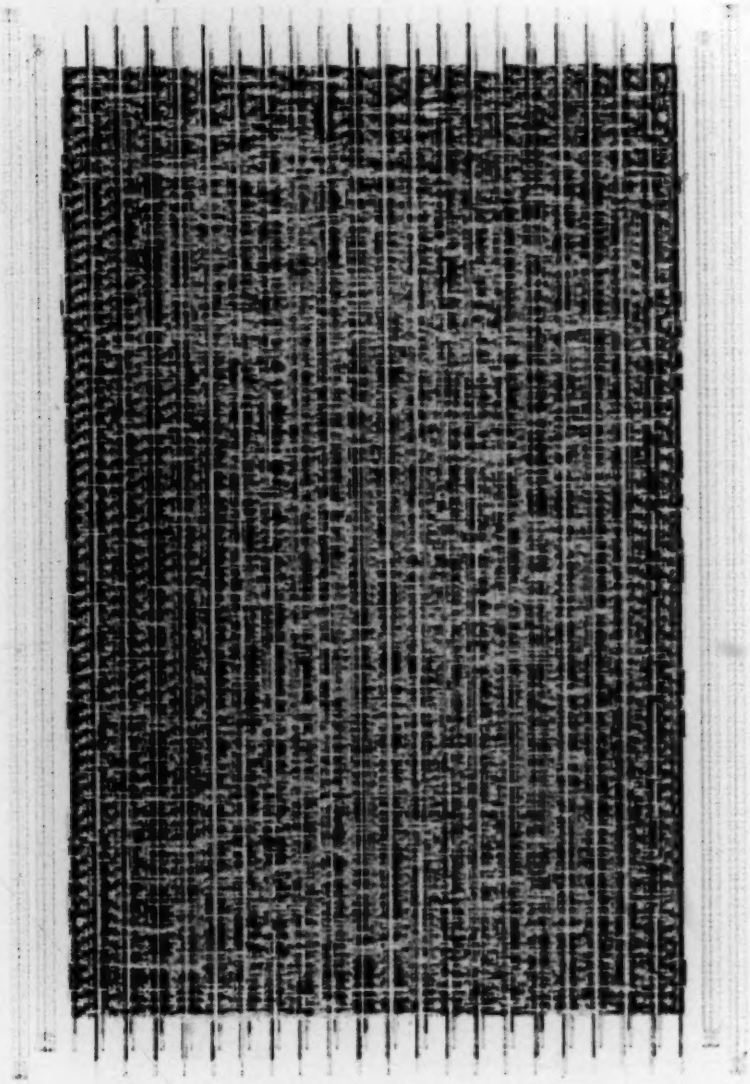
[82] Labrousse et al., "A 50MHz Microprocessor with a Very Long Instruction Word Architecture", ISSCC Dig. Tech. Papers 1990, p. 44-45.

[83] Miyaka et al., "A 40 MIPS (Peak) 64-bit Microprocessor with One-Clock Physical Cache Load/Store", ISSCC Dig. Tech. Papers 1990, p. 42-43.

[84] Richard Foss, Personal Communications, 1990

APPENDIX A: ECC HARDWARE

A.1 (137,128) ECC Hardware



APPENDIX B: HAMMING CODES

B.1 (22,16) IBM System/3 [26]

11111000 10010100 000001
11100111 01000010 000010
00011111 00101001 000100
10010100 11111000 001000
01000010 11100111 010000
00101001 00011111 100000

B.2 (40,32) IBM 8130 [26]

10101010 10101010 11000000 11000000 00000001
01010101 01010101 00110000 00110000 00000010
11111111 00000000 00001100 00001100 00000100
00000000 11111111 00000011 00000011 00001000
11000000 11000000 11111111 00000000 00010000
00110000 00110000 00000000 11111111 00100000
00001100 00001100 10101010 10101010 01000000
00000011 00000011 01010101 01010101 10000000

B.3 (72,64) IBM 3033 [26]

11111111 11111111 00010001 00010001 00100010 00100010 00010001 00010100 00000001
10001000 10001000 11111111 11111111 00010001 00010001 10001000 10000010 00000010
01000100 01000100 10001000 10001000 11111111 11111111 01000100 01000001 00000100
00100010 00100010 01000100 01000100 10001000 10001000 00100010 00101111 00001000
00010001 00010001 00110011 00110011 01110111 01110111 00000000 00001111 00010000
00001111 00001111 00001111 00001111 00001111 00001111 11111111 00000000 00100000
00000000 11111111 00000000 11111111 00000000 11111111 11110000 11111111 01000000
11110000 00001111 11100001 00011110 11000011 00111100 00001111 11110111 10000000

B.4 (72,64) IBM 3081 [25]

11111111 00000000 00000000 11111111 10001110 10001110 10001110 10001110 00000001
11111111 11111111 00000000 00000000 01001101 01001101 01001101 01001101 00000010
00000000 11111111 11111111 00000000 00101011 00101011 00101011 00101011 00000100
00000000 00000000 11111111 11111111 00010111 00010111 00010111 00010111 00001000
10001110 10001110 10001110 10001110 11111111 00000000 00000000 11111111 00010000
01001101 01001101 01001101 01001101 11111111 11111111 00000000 00000000 00100000
00101011 00101011 00101011 00101011 00000000 11111111 11111111 00000000 01000000
00010111 00010111 00010111 00010111 00000000 00000000 11111111 11111111 10000000

B.5 (137,128) IBM DRAM [2]